VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

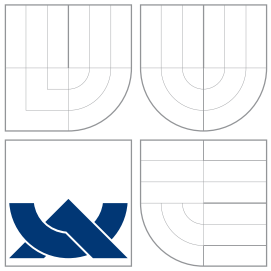
DETEKCE DUPLICITNÍHO PROVOZU

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

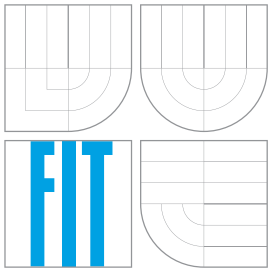
AUTOR PRÁCE
AUTHOR

Bc. PETR KRCH

BRNO 2013



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

DETEKCE DUPLICITNÍHO PROVOZU

DUPLICATE TRAFFIC DETECTION

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. PETR KRCH

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. MATĚJ GRÉGR

BRNO 2013

Abstrakt

Tato práce se zabývá metodami pro detekci duplicitního provozu v počítáčových sítích. Nejdříve je rozebrána problematika redundantního provozu včetně způsobu jeho vzniku a příčin. Dále se práce zabývá na základě jakých dat a jakými metodami jsme schopni duplicitní provoz odhalit. V neposlední řadě se touto studií zabýváme otázkou návrhu algortimu na detekci redundantce včetně návrhu testování a zhodnocením dosažených výsledků.

Abstract

This thesis describes methods for duplicate traffic detection in computer networks. At first, it analyzes the problem of redundant traffic including the description of its origin and causes. The thesis describes on what data and what methods we are able to detect an operation as duplicate. Finally, this thesis explains design of redudant traffic detection algorithm including testing plan and results evaluation.

Klíčová slova

Duplicitní provoz, redundantní provoz, Rabin Fingerprint, Simhash, Redis, Berkeley DB

Keywords

Duplicate traffic, redundant traffic, Rabin Fingerprint, Simhash, Redis, Berkeley DB

Citace

Petr Krch: Detekce duplicitního provozu, diplomová práce, Brno, FIT VUT v Brně, 2013

Detekce duplicitního provozu

Prohlášení

Prohlašuji, že jsem tento semestrální projekt vypracoval samostatně pod vedením pana Ing. Matěje Grégra

.....
Petr Krch
18. května 2013

Poděkování

Na tomto místě bych rád poděkoval Ing. Matěji Grégrovi, jehož pomoc a připomínky pro mne byly motivující a často byly klíčové pro dokončení této práce. Dále bych rád poděkoval své rodině za podporu, díky které mi bylo umožněno se dostat až k psaní této práce. Z celého srdce bych také rád poděkoval všem ze svého nejbližšího okolí za podporu, která mě motivuje a pohání kupředu.
Vám všem upřímně děkuji!

Obsah

Kapitola 1

Úvod

V dnešní době internetu, kdy neustále dochází k navyšování datových toků, se nabízí otázka, do jaké míry se v běžném provozu přenášejí duplicitní data. Tato práce se bude zabývat právě touto otázkou a technikami vedoucími k odpovědi na ní.

V úvodu této technické zprávy samotnou práci představujeme z širšího hlediska a představujeme co bude popsáno v následujících kapitolách. V kapitole 2 se zabýváme co to je vlastně duplicitní provoz a také možnostmi jeho vzniku a jeho příčinami. Dále se v této kapitole zabýváme současnými trendy datového provozu, respektive poměrem přenášených služeb v počítačových sítích. Také si zde definujeme cíle této práce a jasně si stanovujeme její mantinely. Kapitola 3 se poté zabývá možnostmi detekce duplicitního provozu na základě položek jednotlivých protokolů. Tato kapitola nás dovede k odpovědi jaká data provozu pro nás budou určující při detekci redundantního provozu a jaká budeme při analýze ignorovat. V kapitole 4 si blíže rozebereme několik metod vhodných pro detekci duplicity. V této kapitole bude představeno hned několik metod včetně jejich výhod a nevýhod. Zároveň je v kapitole představena námi nově navržená metoda. Kapitola 5, nazvaná Návrh algoritmu, obsahuje volbu metod nejvhodnějších pro detekci redundance a následně i jednoduchý návrh celého algoritmu. Kapitola 6 je vyhrazena pro návrh testování, které budeme na aplikaci provádět. V této kapitole blíže představíme zvolenou sadu testů, kterým byla aplikace podrobena. V následující kapitole 7 si popíšeme průběh implementace analyzátoru a to zejména z pohledu tvorby jednotlivých prototypů a jak nás tyto prototypy ovlivňovali v dalším životním cyklu tohoto projektu. V kapitole 8 zhodnotíme výsledky sad testů na implemntované metody. Asi stěžejní kapitolou této práce je kapitola 9 kde se zamyslíme nad výsledky, které je možno od budoucí analýzy provozu očekávat a dále shrneme statistiky analyzovaného provozu a výsledky analýz při použití různých metod detekce duplicity. V závěrečné 10. kapitole práce je vyhodnoceno, zda bylo splněno zadání a je zde také navržen budoucí vývoj tohoto projektu.

Kapitola 2

Duplicitní provoz

Koncový uživatelé internetu využívají širokou škálu dat napříč celým internetem a to včetně HTTP obsahu, streamovaného videa a audia, VoIP komunikace, herních dat, sdílení souborů v peer-to-peer komunikaci atd. Faktory jako je podobnost v oblasti sociálních zájmů a koníčků, zeměpisné blízkosti, a samozřejmě také popularita některých dat znamená, že je vysoká pravděpodobnost podobnosti přenášených dat ke koncovým uživatelům takové sítě. Některá část přenášeného provozu se tak může překrývat s provozem, který již nastal v minulosti (např. opakovaný přístup na wikipedia.org) a nebo s přenosem od různých hostitelů (např. kopírování článků na více zpravodajských serverů). Větší pravděpodobnost takovéhoto přenášení duplicitního obsahu je blíže u koncových zařízení sítě nebo u menších poskytovatelů internetu, neboť v takových případech tyto skupiny uživatelů a serverů sdílejí kromě síťové infrastruktury i další věci jako například práci, koníčky a zájem o místní události.[2]

V této práci bychom rádi odpověděli na následující otázky: Jak velký poměr datového provozu je duplicitní? Jaké jsou nejčastější zdroje duplicitního provozu? Jaké služby jsou nejčastějším viníkem redundance? Dále se chceme v práci zaměřit na detekci nejenom skupiny bajtů ze streamu, ale také na duplicitu celého payloadu paketů. Výsledky těchto dvou přístupů detekce by nám mohli poskytnou zajímavá data k porovnání.

V této kapitole by bylo také vhodné vysvětlit co to vlastně duplicitní (nebo-li redundantní) provoz je. Duplicitním provozem rozumíme přenášení dat, která již někdy v minulosti byla přenesena. Nezáleží však na tom, zda-li je duplicitní tok znovu vytvářen mezi stejnými koncovými body či nikoliv.

2.1 Vznik a příčiny redundance

Co se týče vzniku duplicitního provozu, tak je hned několik možných způsobů.

- opětovný přenos dat mezi stejným zdrojem a stejným příjemcem - například opakované zhlédnutí stejného on-line videa jedním uživatelem
- přenos stejných dat mezi dvěma a více příjemci od jednoho zdroje - více uživatelů si přečte na internetu tentýž článek z téhož serveru
- přenos stejných dat z dvou a více zdrojů - více koncových uživatelů si stáhne tentýž obrázek z různých zdrojů

Nejčastějším důvodem bývá redundance na základě popularity a to i přes fakt, že v současnosti ty nejnavštěvovanější servery poskytují služby z několika IP adres (z několika

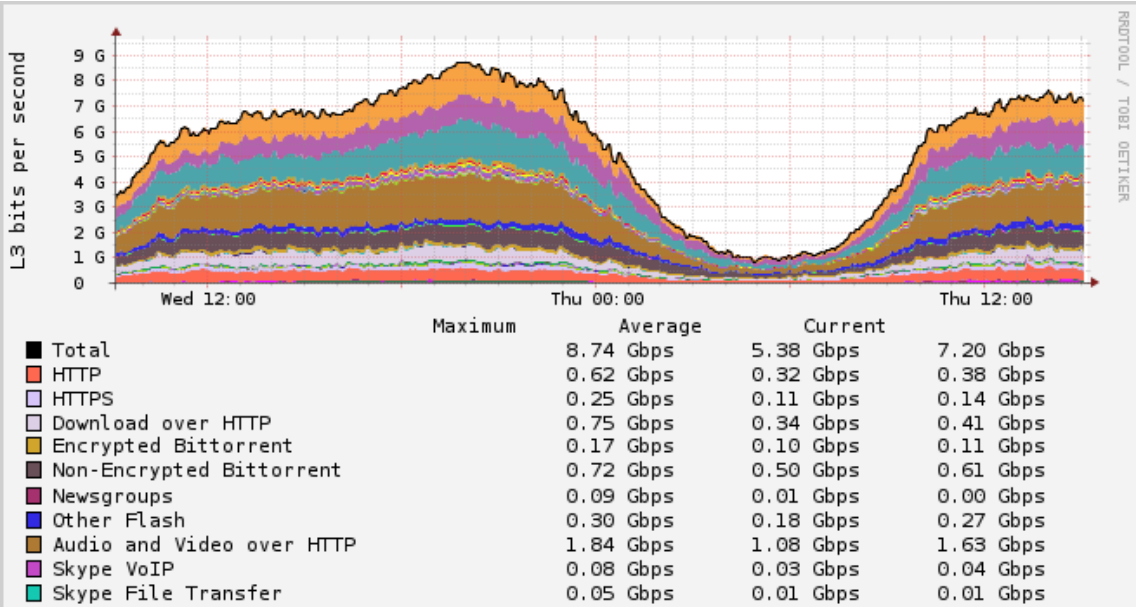
zdrojů). Redundance na základě popularity je přenos od jednoho zdroje k více příjemcům. Tato redundance je navíc v současné době podpořena velkou popularitou sociálních sítí, díky kterým se populární informace šíří velkou rychlostí a tak se výrazně podílejí na velikosti podílu duplicitního provozu.

Dále se nabízí otázka proč jsou totožná data na různých zdrojích? Důvodem může být samozřejmě zálohování těchto dat, mirroring serverů (ten můžeme například vidět na následujícím příkladu překladu doménového jména youtube.com, viz 2.1), ale také prosté kopírování tohoto obsahu (zejména u obrázků je to velmi častý jev). Jako příklad uvádíme výsledek serveru tineye.com (vyhledávač podobných obrázků), který na dotaz nad známým obrazem Leny Söderberg zobrazil 850 výsledků [10] .

Listing 2.1: DNS dotaz nad youtube.com

```
user@pc:~$ host youtube.com
youtube.com has address 173.194.39.137
youtube.com has address 173.194.39.142
youtube.com has address 173.194.39.128
youtube.com has address 173.194.39.129
youtube.com has address 173.194.39.130
youtube.com has address 173.194.39.131
youtube.com has address 173.194.39.132
youtube.com has address 173.194.39.133
youtube.com has address 173.194.39.134
youtube.com has address 173.194.39.135
youtube.com has address 173.194.39.136
```

2.2 Současné trendy datového provozu



Obrázek 2.1: Vybrané významné služby

Datový provoz v současnosti obsahuje velké množství přenášených protokolů. Jak může vypadat takový poměr přenášených služeb můžeme vidět z výše uvedeného grafu (viz graf

2.1) datového provozu (výše jsou uvedeny pouze významné a pro nás podstatné služby, kompletní graf můžeme vidět v příloze C), který je vygenerován zařízením Service Control Engine 8000 (dále jen SCE). Tento graf je ze dne 27.12.2012 a vychází z dat až 34000 aktivních zákazníků komerční společnosti Smart Comp a.s. poskytující službu NETBOX®. Zařízení SCE určuje typ služby nejenom na základě čísla protokolu, ale díky celému obsahu paketu, a proto jsou tyto data co nejpřesnější. Na první pohled je z grafu vidět, že podstatnou část toku stále zabírá HTTP komunikace. Ke grafu je nutno dodat, že zbylé obsáhlé části provozu zabrala komunikace mezi párujícím zařízením. Z tohoto grafu dále vyplývá, že tato komunikace zabírá v průměru něco přes 40 procent provozu. Dále je vidět, že nejpodstatnější část v dnešní době zabírá audio a video přenášené právě přes HTTP. Z grafu vyplývá, že podstatnou část provozu tvoří HTTP komunikace a to včetně právě zmíněného přenášení audia a videa. Tato komunikace bude v naší studii hrát velmi zásadní význam neboť předpokládáme největší zjištění poměru redundance právě v těchto službách. V České republice je například v službě poskytování on-line videa přes HTTP jednoznačně největším zdrojem webová stránka youtube.com . Pro věští ilustraci poměru této stránky na globálním trhu poslouží následující úryvek ze statistiky této služby říkající „V roce 2011 měla služba YouTube více než 1 bilion zhlédnutí. To je 140 zhlédnutí na každého člověka na světě.“ [11]. Další podstatnou službou, která může velmi výrazně ovlivnit množství redundance, zajišťují servery poskytující možnosti sdílení souborů. Tyto servery jsou v současnosti velmi populární a to zejména díky jejich tolerování sdílení tzv. warez (nebo-li dat se kterými je nakládáno v rozporu s autorským právem). V naší republice je jedničkou v tomto odvětví internetu služba uloz.to. Je tedy namístě předpokládat, že tyto servery budou jedním z klíčových zdrojů duplicitního provozu.

2.3 Šifrováný provoz

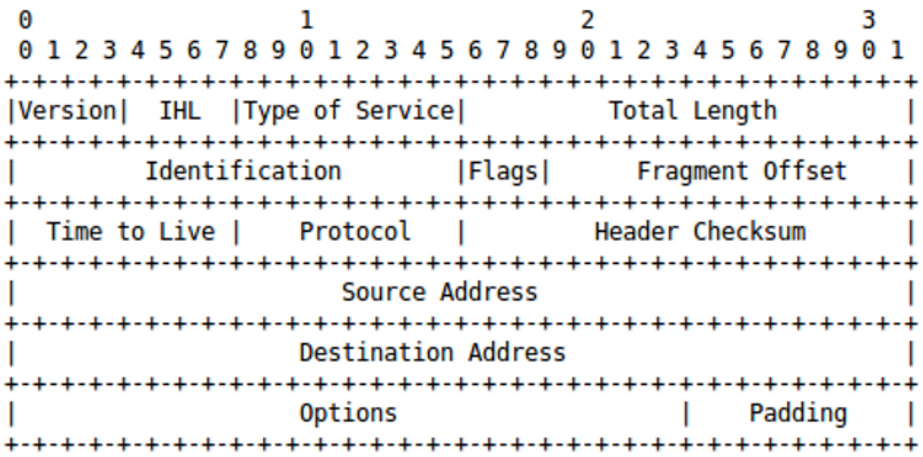
V dnešní době je čím dál větší datový provoz šifrován (z výše uvedeného grafu vidíme, že například poměr HTTPS ku HTTP protokolu je cca 2:5). Jak bychom však mohli duplicitní šifrovaný provoz odhalit? Šifrovaný provoz bychom mohli detekovat pouze na základě několika vedlejších parametrů přenášených dat (jako například jejich velikosti a jejich zdrojem). V takovém případě bychom ale nedosáhli korektních výsledků detekce takovéto komunikace. Ty by byly zkresleny několika faktory. Například bychom nemohli splnit podmínku detekce duplicitního provozu z několika různých zdrojů. Navíc by docházelo k velkému počtu False Positive případů detekce. Jelikož se chceme dostat k co nejpřesnějším výsledkům při naší analýze, tak jsme se rozhodli konstatovat, že se otázkou duplicity šifrovaného provozu nebudeme v této studii zabývat.

Kapitola 3

Identifikace paketu

Abychom dokázali detekovat duplicitní provoz je potřeba určit, které části provozu jsou nejvíce určujícím faktorem redundance. Z tohoto důvodu je nutné se blíže podívat na přenášená data včetně dat určených k režii spojení a dále tyto údaje analyzovat z hlediska použitelnosti v našem projektu. Při této analýze je potřeba brát v úvahu již provedený rozbor vzniku duplicitního provozu, tak abychom pokryli všechny možnosti redundance. V následujících podkapitolách si rozebereme různé možnosti detekce duplicitního provozu a to včetně jejich výhod i nevýhod.

3.1 IP hlavička



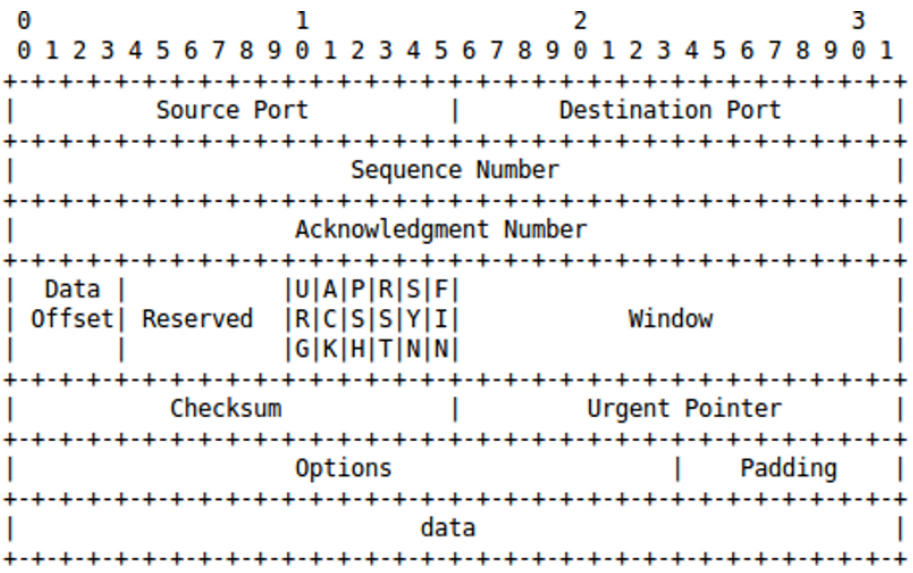
Obrázek 3.1: IP hlavička [5]

Jako první si rozebereme možnost detekce redundance na základě IP hlavičky. Prvně je potřeba říci, že taková detekce je skutečně možná. Pokud se podíváme na jednotlivé složky IP datagramu (znázorněné na obrázku 3.1), zjistíme, že z IP hlavičky jsme schopni detekce na základě zdrojové či cílové IP adresy. Příkladem takového provozu může být jakýkoliv přenos většího množství dat z jednoho zdroje. Každý paket komunikace by pak byl (na základě IP adres) vyhodnocen jako duplicitní a to i v případě přenášení zcela jiných dat.

Takováto detekce má však velké množství nevýhod. Zejména je nutné zdůraznit, že díky této detekci bude velké množství False Positive i False Negative výsledků. False Positive detekce budou způsobeny jakýmkoliv provozem, bez ohledu na přenášený obsah, ze zdrojové IP adresy (na této adrese se může navíc nacházet velké množství poskytovaných služeb). False Negative pak budou ty případy, kdy jsou tatáž data přenášena z více zdrojů a to ať díky mirrorování nebo kopírování obsahu (oba tyto případy jsou blíže popsány v kapitole 2). V našem projektu se musíme tedy naopak snažit odprostit od zdrojové i cílové adresy. K výhodám této detekce patří její jednoduchost. O tu se nám ale v tomto projektu nejedná, a proto se na žádnou z položek IP datagramu nemůžeme spoléhat, avšak položky z IP hlavičky mohou sloužit k doplnění dalších dat. Z výše uvedených důvodů ale spíše pro tvorbu statistik celkové analýzy než pro detekci duplicity. Z tohoto důvodu se blíže zaměříme na protokoly vyšších vrstev a pokusíme se potřebné položky, které by byly vhodné pro detekci duplicitního provozu nalézt tam.

3.2 TCP hlavička

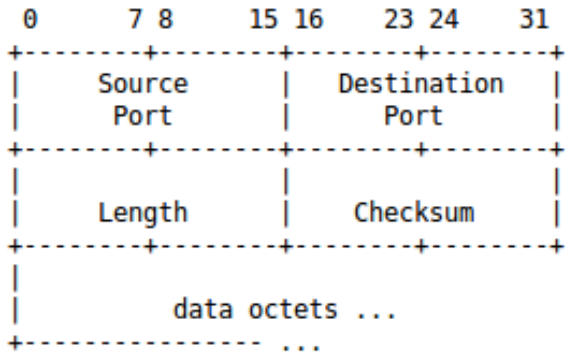
Nyní, když nebereme v úvahu zdrojovou a cílovou adresu komunikace, respektive pominemeli celou IP hlavičku, tak se můžeme blíže zaměřit na položky z TCP formátu (viz obrázek 3.2) díky kterým bychom mohli detekovat redundantní provoz. Ten jsme schopni z těchto dat samozřejmě také detekovat. V tomto případě, jak si však vzápětí ukážeme, již s mnohem lepší přesností a také v celé šířce možných redundancí. Z formátu TCP hlavičky jsou pro nás nejzajímavější položky Source Port, Destination Port a pak samotná data. Ze zdrojového nebo cílového čísla portu jsme schopni ve většině případů detekovat přenášenou službu. Toto ovšem například neplatí při použití NAT (Network address translation). Navíc (i pokud pomineme tuto skutečnost) bychom se dostali do podobného problému jako v případě detekce pomocí IP adres v předešlé kapitole. V tomto případě by pak byla jako duplicitní provoz vyhodnocena například i komunikace více uživatelů se stejnou službou téhož zdroje. A to opět nezávisle na přenášených datech. Ani v případě detekce za použití kombinace IP adres a použitých portů bychom nedospěli v potřebnému zpřesnění analýzy. Proto je potřeba se pro identifikaci redundance zaměřit na samotnou datovou část paketu.



Obrázek 3.2: TCP hlavička [6]

3.3 UDP hlavička

Náš pohled na detekci UDP datagramu (obrázek 3.3) bude, z hlediska významů položek v něm obsažených, totožný jako pohled na položky v TCP paketu. Stejně jako u něj se nemůžeme spoléhat na čísla portů a proto se budeme zaměřovat na datovou část paketu. Navíc je tento protokol, díky nespojované službě, využíván zejména službami, které nepatří mezi potencionální zdroje duplicitního provozu. Jakým stylem bude detekce duplicity dat probíhat se podíváme v následující kapitole.



Obrázek 3.3: UDP hlavička [4]

Kapitola 4

Otisk paketu

Nyní již víme, které části paketu jsou pro nás zajímavé. Pro detekci v našem analyzátoru budeme využívat výhradně přenášených dat. Tato data ale bude vhodné dále upravit a udělat jakýsi jejich otisk (v případě některých metod hned několik otisků), který je bude co nejjednoznačněji identifikovat. Tento krok je nutný zejména díky rychlosti vyhledávání v datech z již proběhlé analýzy provozu.

4.1 Rabin fingerprint

Metoda Rabin fingerprint byla prezentována Michaelem O. Rabinem v roce 1981. Tato metoda spočívá v rozdělení každého streamu na sekvence o velikosti β bajtů. Následně se pro tyto oddíly bajtů $t_1, t_2, t_3, ...t_\beta$ provede výpočet podle níže uvedeného vzorce, ve kterém p (prvočíslo) a M ($M = 2^n$ a $n\epsilon$ $\mathbb{N}$) jsou konstanty:

$$RF(t_1, t_2, t_3, ...t_\beta) = (t_1 p^{\beta-1} + t_2 p^{\beta-2} + ... + t_{\beta-1} p + t_\beta) \mod M \tag{4.1}$$

Je nutné doplnit, že v citovaném textu (viz [9]) je uveden níže uvedený vzorec 4.2. Avšak jak je vidět při důkladnějším pohledu, tak takto indexované proměnné nemohou vést k požadovanému výpočtu. Máme-li v tomto vzorci indexaci proměnné t od 1 do β, pak není možné u proměnné p používat mocniny od β do 0. Patrně se jednalo pouze o překlep při psaní tohoto článku, ale i tak bylo potřeba větší pozornosti při další práci s tímto textem.

$$RF(t_1, t_2, t_3, ...t_\beta) = (t_1 p^{\beta} + t_2 p^{\beta-1} + ... + t_{\beta-1} p + t_\beta) \mod M \tag{4.2}$$

Tento vzorec (4.1) vytváří pro všechny skupiny bajtů $\{(t_1, t_2, ...t_\beta), (t_2, t_3, ...t_{\beta+1}), ...\}$ velikosti β jednoznačný otisk. Výpočet dalších otisků se dá navíc odvodit od předešlého kroku a to celý výpočet značně zrychluje. Výpočet následujících kroků se pak řídí dle následujícího vzorce 4.3.

$$RF(t_{i+1}...t_{\beta+i}) = (RF(t_i...t_{\beta+i-1}) - t_i * p^{\beta-1}) * p + t_{\beta+i} \mod M \tag{4.3}$$

Pro zrychlení výpočtu je možno také navíc pro výraz $t_i * p^\beta$ předpočítat tabulku. Tato tabulka bude obsahovat pouze 256 záznamů jelikož p a β jsou konstanty a t_i je velikosti jednoho bajtu. Tento výraz pak může nabývat pouze 256 hodnot.[9]

4.1.1 Příklad výpočtu

Pro větší ilustraci tohoto algoritmu provedeme názorný příklad výpočtu dle výše uvedených vzorců. Před samotným výpočtem je potřeba si vhodně zvolit konstanty v tomto vzorci. Konstanta p musí být prvočíslo a to dostatečně veliké, aby měl algoritmus požadovaně velkou rozlišovací schopnost. M je číslo, které udává velikost množiny možných výstupů hashovací funkce, proto je toto číslo potřeba zvolit dostatečně veliké abychom minimalizovali pravděpodobnost kolize, ale zároveň měli na paměti výpočetní kapacity detekčního zařízení. Velikost β udává pro jak velké bloky bajtů se bude hash vypočítávat. Tedy pro jak velké bloky bajtů budeme vyhledávat duplicitu. Pokud bychom zvolili β příliš malé docházelo by k umělému navyšování výsledků redundantního provozu. V opačném případě bychom ztratili výhodu tohoto algoritmu v překrývání jednotlivých bloků.

Máme-li vstupní řetězec „sarkmjrasjdsjfaskjfcscweiwekjekxy“ (pro názornost uvádíme řetězec, kde jednotlivé znaky znázorňují číselnou hodnotu velikosti jednoho bajt) a zvolíme-li si konstanty $\beta = 32, M = 2^{60}, p = 436507$ pak výpočet podle metody Rabin fingerprint bude následující:

$$
\begin{gathered}
t_1 = s = 115 \\
t_2 = a = 97 \\
... \\
t_{31} = k = 107 \\
t_{32} = x = 120 \\
t_{33} = y = 121
\end{gathered}
$$

$$
\begin{gathered}
RF(t_1...t_{32}) = RF(sarkmjrasjdsjfaskjfcscweiwekjekx) = \\
= (115 * 436507^{32} + 97 * 436507^{31} + ... + 107 * 436507 + 120) \mod 2^{60}
\end{gathered}
$$

$$
\begin{gathered}
RF(t_2...t_{33}) = RF(arkmjrasjdsjfaskjfcscweiwekjekxy) = \\
(RF(t_1...t_{32}) - 115 * 436507^{32}) * 436507 + 121 \mod 2^{60}
\end{gathered} \tag{4.4}
$$

Výše uvedeným výpočtem 4.4 metodou Rabin fingerprint s nastavenými parametry nad řetězcem „sarkmjrasjdsjfaskjfcscweiwekjekxy“ získáme dva otisky. V případě, že některý další řetězec bude obsahovat totožnou posloupnost velikosti β (v našem případě 32) bajtů, pak výpočtem získáme totožný otisk a můžeme tento řetězec prohlásit za duplicitní.

K výhodám tohoto algoritmu patří zejména jeho pružnost, která je způsobena parametry, které zásadně ovlivňují výstup tohoto algoritmu. K dalším výhodám této metody patří porovnávání bloků jedné velikosti, který se ovšem při výpočtech postupně posouvá a tak

bere v úvahu všechny výskyty podřetězců o předem dané velikosti bloku. Tato výhoda je ale vykoupena značnou nevýhodou tohoto algoritmu a tou je velké množství vygenerovaných hashí.

4.2 Simhash

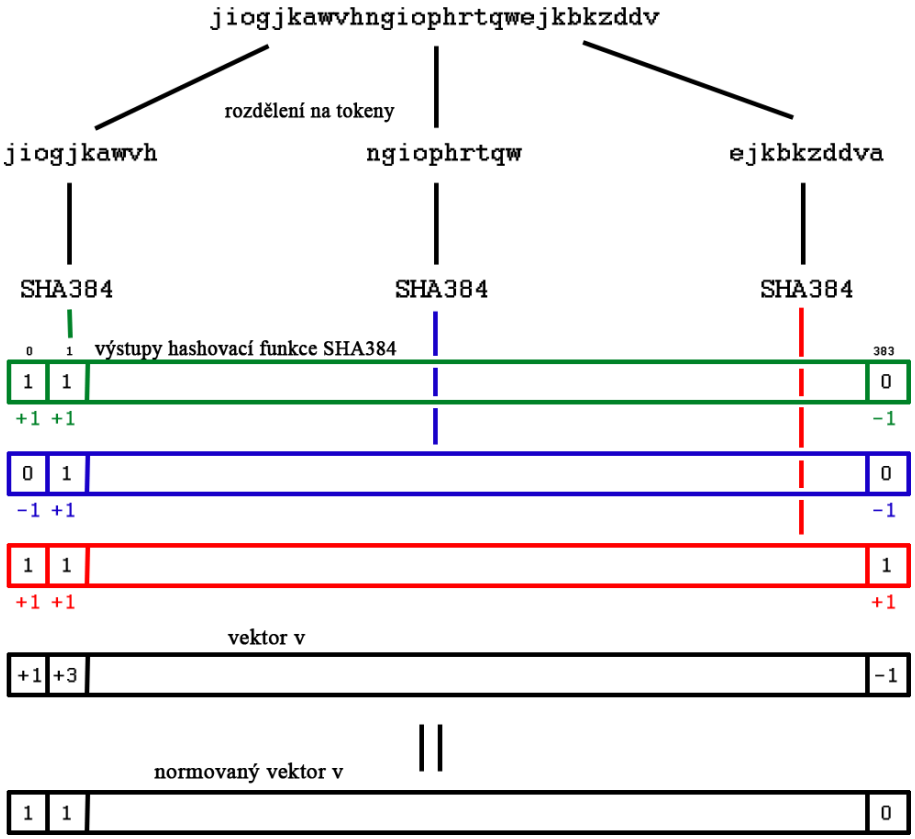
Tato metoda spočívá v rozdělení řetězce na tzv. tokeny, pro které je vytvořen výstup hashovací funkce (pro každý token je vygenerován samostatný hash), který je dále zpracováván. Tento algoritmus je níže popsán detailněji v jednotlivých krocích (včetně názorného příkladu výpočtu na obrázku 4.1). Výstupem této metody je vektor předem dané velikosti (velikost je určena na základě velikosti výstupu použité hashovací funkce), který může nabývat pouze hodnot 0 a 1. Tato metoda při porovnávání duplicity neuvažuje pořadí jednotlivých tokenů, ale pouze jejich výskyt. Na rozdíl od předchozí metody také nebere v úvahu podobnost jiných podřetězců než právě vybraných tokenů. Respektive nedochází k porovnávání dat která se překrývají mezi tokeny. V našem případě bychom pro tuto metodu zvolili konstantní velikost tokenu, která by odpovídala velikosti β u algoritmu Rabin Fingerprint.

Slovní popis algoritmu simhash [8]

1. Datovou část paketu rozdělíme do bloků (tokenů).
2. Všechny dimenze vektoru v (který má stejnou velikost jako počet bitů výstupu použité hashovací funkce) jsou inicializovány na 0.
3. Pro každý takto vytvořený blok generujeme otisk stejnou hashovací funkcí.
4. Pro každý bit výstupu hashovací funkce pak provedeme kontrolu. Pokud je tento bit (na pozici k) v logické 0 provede se snížení dimenze vektoru (na pozici k) o 1. V opačném případě se dimenze o 1 zvýší. Tato kontrola se provede pro každý výstupní bit výstupu hashovací funkce generovaného nad každým tokenem.
5. Jako další krok celý vektor v normujeme. A to tak, že pokud dimenze vektoru je větší nebo rovna 0, tak bude její hodnota nastavena na 1. V opačném případě bude hodnota 0. Toto provedeme pro každou dimenzi vektoru v
6. Nyní máme pro tuto datovou část paketu vygenerován hash ve vektoru v

4.2.1 Výpočet podobnosti

V případě určení podobnosti dvou vektorů generovaných algoritmem Simhash můžeme provést výpočtem jejich Kosinovy podobnosti a nebo pomocí Hammingovy vzdálenosti dvou porovnávaných vektorů. V případě použití Kosinovi podobnosti pak podobnost vektorů x a y vypočítáme podle vzorce 4.5 [1].



Obrázek 4.1: Algoritmus Simhash [6]

$$Kosinova\_podobnost(x,y) = \frac{\sum_i x[i] \cdot y[i]}{\sqrt{\sum_i x[i]^2} \times \sqrt{\sum_i y[i]^2}} \tag{4.5}$$

Pokud budeme podobnost počítat pomocí Hammingovi vzdálenosti, pak bude výpočet podobnosti vektorů x a y podle následujícího vzorce 4.6.

$$Hammingova\_vzdalenost(x,y) = \sum_i |x[i] - y[i]| \tag{4.6}$$

V našem případě analýzy bychom tedy vždy po výpočtu provedli výpočty podobností s předešlými pakety. Celkově bychom tedy u n paketů provedli x výpočtů podobnosti, kde x vypočítáme podle následujícího vzorce 4.7.

$$x = \frac{\binom{n}{2}}{2} \tag{4.7}$$

U nevýhod metody Simhash bychom se (v případě našeho použití) zmínil o faktu, že tato metoda nebere v úvahu pořadí jednotlivých tokenů, ale pouze jejich výskyt. K dalším nevýhodám této metody (oproti metodě Rabin Fingerprint a z hlediska použití v našem projektu) patří fakt, že nedochází ke zpracování bloků napříč tokeny čímž mohou být výsledky negativně ovlivněny.

4.3 Hashovací funkce MD5

V citovaném článku „A protocol-independent technique for eliminating redundant network traffic“ [9] byla metoda použití hashovací funkce MD5 popsána jako nevhodná a to bez podkladů pro toto tvrzení. My bychom ovšem rádi provedli testy nejenom na duplicitu streamů, ale také duplicitní provoz samotných paketů. Pro tuto statistiku se použití této metody nejevilo jako zcela nereálné, a proto jsme tuto metodu podrobili několika testům. Z těchto testů jsme získali pozitivní výsledky a tak jsme mohli tuto metodu prohlásit za vhodnou pro zařazení k tvorbě prototypu nad kterým budou provedeny další série testů. Při této metodě využíváme MD5 hashe každé datové části každého paketu. Díky této hashovací funkci vytvoříme pro každou datovou část hash, která bude představovat index těchto dat. Tento index je pak jednoznačným (bereme-li v úvahu ideální hashovací funkci) otiskem paketu. Pak tedy v případě shody více hashí můžeme data z kterých byla hash vypočítána prohlásit za duplicitní.
K výhodám této metody patři rychlost. Analýza touto metodou by měla probíhat o poznání rychleji, avšak z uvedených metod je na podmínky uznání redundance nejpřísnější, neboť se musí shodovat celá datová část paketu. K jednoznačným nevýhodám patří nepřesné výsledky. Neboť například pakety stahované různými prohlížeči se v datové části mohou lišit.

4.4 Metoda MD5 token

Tato metoda byla vytvořena jednoduchou modifikací metody MD5 popsanou výše. V této metodě provedeme rozdělení datové části paketu na tokeny o velikosti β a nad těmito tokeny pak následně provedeme generování hashe pomocí funkce MD5. Generování provádíme pouze nad tokeny o velikosti β, takže případný zbytek po dělelní β na konci paketu ignorujeme.

4.5 Rabin Fingerprint Offset Search

K návrhu této metody jsme byli inspirováni po vytvoření jednoho z prototypů. Ten nám již v praxi ukázal, že metoda Rabin Fingerprint produkuje opravdu velké množství hashí. Tento počet je možné (do jisté míry) zredukovat nastavenými parametry. Ovšem ke značnému snížení počtu hashí dojde až v případě parametrů nevhodných pro detekci. Z těchto důvodů jsme si dovolili tuto metodu určitým způsobem modifikovat. Modifikace spočívá zejména ve zvýšení kroku o který se budeme při počítání hashí posouvat. U klasické metody Rabin Fingerprint dochází byte po bytu. V modifikované verzi bychom se chtěli posouvat o celý tento blok, tedy krok mezi bloky by byl o velikosti β. V takovém případě bychom sice nezachovali překrývání takto vytvořených tokenů, čímž jistě ztratíme jednu z výhod klasického algoritmu. Tedy fakt že jsme schopni zachytit všechny posloupnosti velikosti β.

Tuto ztrátu bychom rádi eliminovali výpočtem vhodného startovacího bytu od kterého se budeme posouvat. Tuto pozici bychom vypočetli z několika vypočítaných hashí, které by sloužily pouze k nalezení nejvhodnějšího offsetu od kterého se budeme posouvat napříč celou datovou částí paketu. Ve fázi hledání správné startovací pozice si navíc ověříme u sousedního tokenu zda jsme nalezli správnou pozici. V případě, že žádná pozice nevyhovuje námi popsaným podmínkám, pak budeme generovat hashe nad tokeny od začátku paketu. Jsme-li tedy ve fázi, kdy známe počáteční pozici od které budeme tokeny posouvat, tak pro celý paket vygenerujeme hashe nad tokeny o velikosti β, které postupně posouváme o velikost β. Generování provádíme pouze nad tokeny o velikosti β, pokud tedy velikost paketu není dělitelná β, tak u některých dat nedojde k vygenerování hashe. K této volbě jsme se přiklonili z důvodu eliminace False Positive výsledků, které by mohli menší tokeny způsobovat. V následujícím odstavci si tento algoritmus popíšeme přehledně v jednotlivých krocích.

Slovní popis algoritmu Rabin Fingerprint Offset Search - fáze nalezení offsetu

1. Je datová část paketu větší alespoň jako $3 \times \beta$? Pokud ano pokračuj na bod 2, jinak ukonči hledání offsetu s výsledkem 0 (tedy bez posunu).
2. Z konce datové části se posunem krokem o velikosti $3 \times \beta$ směrem k začátku a nastav proměnou offset na 0.
3. Je proměnná offset větší nebo rovna β? Pokud ano, tak nastav proměnou offset na 0 a ukonči fázi hledání. Jinak pokračuj následujícím bodem.
4. Z nastavené pozice vypočteme hash z tokenu o velikosti β.
5. Existuje již tento hash z předešlých výpočtů? Pokud ano tak pokračuj na bod 6. V opačném případě pokračuj následujícím bodem.
6. Pokud si již provedl výpočet β hashí, tak pokračuj na bod . Jinak proveď shift pozice tokenu směrem ke konci o jeden byte a pokračuj bodem 3.
7. Proveď kontrolu sousedního tokenu (tedy tokenu s posunem $\pm\beta$). Pokud i ten již existuje, tak jsme nalezli požadovaný offset. Ukonči tedy fázi hledání. Pokud hashe nad sousedním tokenem neexistuje, tak inkrementuj offset, proveď posun tokenu o jeden krok směrem ke konci paketu a pokračuj bodem 3.

Nyní jsme ve fázi, kdy známe offset a tak můžeme pokračovat k fázi generování jednotlivých hashí tohoto paketu.

Slovní popis algoritmu Rabin Fingerprint Offset Search - fáze generování a hledání hashe

1. Dle získaného offsetu, který udává posun od konce si pro přehlednost přepočítáme na offset od začátku datové části paketu (pro zrychlení algoritmu je možnost posouvat tokeny i od konce). Nastavíme pozici prvního tokenu dle proměnné offset.
2. Provedeme výpočet hashe nad tímto tokenem.

3. Existuje tato hash ve dříve vygenerovaných hashích? Pokud ano, tak data nad tímto tokenem o velikosti β jsou duplicitní. Jinak ulož hash do databáze pro pozdější zpracování.

4. V dalším kroku posuneme token směrem ke konci paketu o krok o velikosti β. V případě, že konec tokenu je již posunut mimo data paketu, tak algoritmus ukonči.

Pro přehled ještě zobrazíme algoritmus 4.1, kterým po zavolání funkce *get_fingerprints(self, text, db)* , kde v parametru *text* předáváme řetězec nad kterým chceme vypočítat hashe a parametrem *db* předáváme databázi u které využijeme funkční *exists()*, která vrací pravdu pokud *hash* předaná v parametru v databázi již existuje.

Listing 4.1: Algoritmus Rabin Fingerprint Offset Search

```
# funkce pro nalezeni offsetu diky jiz ziskanym hashs
def searchOffset(self, text, db):
  self.offset = 0 # inicializace offsetu
  if(self.text_len < (3 * self.beta) ):
    # v pripade ze zpracovavany text je mensi nez 3*beta, tak vrat offset 0
    return

  # v teto promenne ukladame posun od konce textu
  # pri predavani vysledku offset prepocitame na posun od zacatku textu
  tmp_offset = 0

  # pro interval celych cisel <0,32); pro postupny posun
  for tmp_offset in range(0 , self.beta):
    # generuj hash pro token o velikosti beta zacinajici na pozici 3*beta +
      offset od konce textu
    hash = self.fingerprint(text[(self.text_len - 3 * self.beta) +
      tmp_offset : (self.text_len - 2 * self.beta) + tmp_offset])
    if(db.exists(hash)):
      # prepocitame na posun od zacatku text a provedeme ulozeni offsetu
      self.offset = (self.text_len + tmp_offset) % self.beta
      # exituje-li tento hash, pak zkontrolujeme sousedni (nasledujici)
        token
      hash = self.fingerprint(text[(self.text_len - 2 * self.beta) +
        tmp_offset : (self.text_len - 1 * self.beta) + tmp_offset])
      if(db.exists(hash)):
        # v pripade, ze i sousedni hash existuje, pak ukonci funkci a vrat
          nalezeny offset
        return

# funkce vracejici seznam vypocitanych hashi dle nalezeneho offsetu
def fingerprints(self, text, db):
  self.text_len = len(text) # velikost zpracovavaneho textu

  # volani funkce pro nalezeni offsetu
  self.searchOffset(text, db)

  fingerprints=[] # seznam vyslednych hash
  if (self.text_len <= self.beta):
    # je-li text mensi nez definovane beta, pak vypocti pouze jednu hash nad
      celym textem
    fingerprints.append(self.fingerprint(text))
```

```
  else:
    # v opacnem pripade vypocitej hash pro tokeny s posunutim dle promenne
        offset
    tokens=[text[i:i+self.beta] for i in range(self.offset, self.text_len,
        self.beta)]
    for t in tokens:
      if (len(t) == self.beta):
        # overeni zda generujeme hash pro uplny token
        fingerprints.append(self.fingerprint(t))
  return fingerprints

# generovani hash dle algoritmu Rabin Fingerprint
def fingerprint(self, token):
  sum = 0
  # pro interval celych cisel <0,32)
  for i in range(0, len(token)):
    sum += ord(token[i]) * pow(self.prime, (self.beta - i) - 1)
  result = sum % self.modulo
  return result
```

K výhodám tohoto algoritmu by mělo patřit relativně malá velikost ukládaných dat. Respektive množství generovaných hashí nad pakety. K nevýhodám bychom přiřadili patrně nutnosti počítání několika hashí, které ve výsledku nebudou nijak uloženy. Tato nevýhoda je však minimalizována faktem, že tento výpočet ve výsledku analýzu prodlouží jen minimálně.

Kapitola 5

Návrh algoritmu

Na základě již sepsaného rozboru je potřeba provést návrh algoritmu, který bude použit pro detekci duplicitního provozu. Níže uvedený návod byl vytvořen nezávisle na budoucí volbě jednotlivých použitých technologií, ať už máme na mysli programovací jazyk, systémy pro ukládání dat aj.
Na základě rozboru metod (provedeným v předchozí kapitole) by bylo nyní vhodné rozhodnout, které z popsaných metod k vytváření otisků použijeme v našem algoritmu. My však chceme ukázat právě rozdíly mezi těmito metodami a tak jsme se rozhodli implementovat všechny tyto metody. To, jestli se v následujících krocích tohoto projektu nakonec pro analýzu použijí či nikoliv, by měli ukázat až výsledky prováděných testů.
Pro náš projekt tedy budeme implementovat aplikaci s metodami Rabin Fingerprint, Simhash, MD5, MD5 token a Rabin Fingerprint Offset Search. Naše aplikace bude moci provádět analýzu pomocí všech těchto metod. Čímž získáme potřebné data pro vyhodnocení výhod a nevýhod těchto testů již po prvních testech. Jak by měl algoritmus vypadat si ukážeme na následujících řádcích. Jako první si představíme pseudokód analyzátoru. V tomto pseudokódu 5.1 budeme ignorovat zpracovávání parametrů, otevírání potřebného vstupního souboru a jiné nepodstatné věci pro znázornění práce programu.

Listing 5.1: Pseudokód analyzátoru

```
main() {
  while(nacti_dalsi_paket) {
    # v parsovani provedeme ziskani informaci z hlavicek
    parsuj_tento_paket

    if (je_paket_sifrovany) {
      continue
    }

    uloz_velikost_dat_v_paketu

    # na zaklade vybrane metody vypocti hash
    hashs = metoda.get_hashs(data_z_paketu)

    # pro vsechny vygenerovane hashe proved nasledujici krok cyklu
    for h in hashs {
      vystup = databaze.inkrementuj_nebo_vloz(h)
      if (vystup == redundance) {
```

```
        uloz_mnozsti_duplicitnich_dat
      }
    }

    redundance = duplicitni_data / celkova_data
}
```

Slovní popis navrhnutého algoritmu

1. Prvním úkolem algoritmu bude rozparsování paketu. Tedy zbavení se (případně uložení) informací přenášených v jednotlivých hlavičkách protokolů druhé, třetí i čtvrté vrstvy ISO/OSI modelu.
2. Dalším krokem algoritmu bude filtrace provozu, kterým jsme se rozhodli v našem projektu nezabývat. Konkrétně máme na mysli filtraci šifrovaného provozu. Tato filtrace bude probíhat na základě čísel portů. Provoz tedy bude filtrován pokud se jedno z čísel portu bude rovna číslu odpovídajícímu jednomu z šifrovaných protokolů.
3. Pokud se v paketu nepřenášejí šifrovaná data, tak si do proměnné uložíme velikost přenášených dat.
4. Zvolenou metodou vygenerujeme nad přenášenými daty hash/e.
5. Pro všechny tyto hashe provedeme kontrolu zda se nejedná o duplicitní provoz. Pokud ano, tak si do proměnné uložíme množství dat, která byla vyhodnocena jako duplicitní.
6. Pokud existuje další paket na analýzu, tak pokračuj bodem 1. V opačném případě přikročíme k následujícímu bodu.
7. Ze získaných proměnných vypočítáme množství duplicitních dat.

Kapitola 6

Návrh testování

Ještě před započetím samotné implementace je potřeba si udělat návrhy testů, které budou na program aplikovány. Pro testování našeho analyzátoru bylo vytvořeno několik záznamů síťového provozu, který byl vytvořen řízeně a u kterého jsme znali procento redundantního provozu. Tyto vytvořené záznamy jsou na přiloženém DVD (viz příloha B). Nad těmito záznamy jsme pak prováděli analýzy za použití všech metod a porovnávali dosažené výsledky s očekávanými. Základním testem aplikace bude analýza záznamu pořízeného na koncovém zařízení uživatele. Na tomto zařízení vytvoříme několik záznamů se simulovaným duplicitním provozem o různém rozsahu a tyto záznamy následně podrobíme analýzou vytvořenou aplikací. Pokud testy v této fázi budou úspěšné přikročíme k fázi další. Během této fáze budeme sledovat provoz několika počítačů, které budou pod naší kontrolou. I v této fázi vytvoříme několik rozlišných záznamů. Dále bude také nutné aby alespoň jeden ze záznamů obsahoval co největší škálu protokolů (máme na mysli veškeré protokoly bez ohledu na to, zda-li u nich bude detekce prováděna či nikoliv). Pokud i výsledky těchto testů budou v pořádku, můžeme přikročit k testům nad objemnými daty pořízenými na páteřním zařízení sítě VUT. Analýzou získaná data budeme pak porovnávat s očekávanými výsledky. V neposlední řadě nám tyto testy ukáží kterou metodou získané výsledky můžeme považovat za nejpřesnější. Případně zda některou z metod prohlásíme za nevhodnou a koncovou analýzu s ní provádět nebudeme.

6.1 Data pro testování

Jak již bylo řečeno data nad kterými jsme prováděli jednotlivé testy byly vytvořeny tak abychom věděli jaké výsledky je možné očekávat od jednotlivých vstupních dat. Níže si blíže rozebereme jednotlivé záznamy a to včetně procent očekávané redundance.

- google_png_2_times.pcap
  V tomto záznamu je zachycen provoz opakovaného stažení obrázku loga společnosti Google témže klientem z téhož serveru a to pomocí nástroje wget (tedy bez využití cachovacích systémů internetových prohlížečů). V tomto případě pochopitelně očekáváme 50% redundanci provozu

- google_png_10_times.pcap
  Tento záznam je vytvořen za stejných podmínek jako předešlí záznam. V tomto případě však bylo logo staženo desetkrát a navíc z třech různých serverů (konkrétně se

jedná o servery s IP 173.194.44.247, 173.194.44.248 a 173.194.44.255). Je tedy očekávána redundance 90%

- idnes_2_times.pcap
  Záznam zachycuje navštívení stránek www.idnes.cz od různých klientů s odlišnými operačními systémy a s různými internetovými prohlížeči. Díky použití různých internetových prohlížečů očekáváme redundanci ne přesně 50%, ale měli bychom se této hodnotě blížit.

- music_from_file_hostings_3_times.pcap
  Stažení téhož hudebního souboru ve formátu *.mp3 z 3 serverů používaných pro sdílení souborů. Konkrétně se jedná o servery www.uloz.to, czshare.com a leteckaposta.cz. U tohoto testu jsou kromě samotného stahování souborů i potřebné úkony vedoucí ke stahování (jako například podrobení se Turingově testům). Navíc jsou soubory stahované z různých internetových prohlížečů. Ideálně by tato analýza měla skončit výsledkem 66,66%, ale díky předešlému důvodu bychom se měli pohybovat okolo 66% celých.

- youtube_2_times.pcap
  Tento provoz byl vytvořen přehráním totožného videa na serveru youtube.com. Video bylo přehráváno na témže zařízení za použití různých prohlížečů. Za ideálních podmínek bychom měli analýzou nad tímto záznamem dospět k výsledku 50%.

- big_one_file_no_redundancy.pcap
  Jedná se o záznam zachycující stahování jednoho objemného souboru. Konkrétně se jedná o soubor (ipv4.download.thinkbroadband.com/100MB.zip) ve formátu *.zip o velikosti 100MB. Tedy síťový provoz bez redundance, proto budeme u analýzy tohoto souboru očekávat výsledek 0% redundance.

Kapitola 7

Implementace

V první fázi implementace byl vytvořen prototyp aplikace na detekci duplicitního provozu. Díky tomuto prototypu jsme odhalili hned několik možných komplikací, kterým jsme se při implementaci finální aplikace mohli vyhnout. Zejména se jednalo o problémy s ukládáním dat. A to zejména co se týče rychlosti a kapacity. Nastávala obvyklá situace, že vyřešením jednoho z problémů vyvstával problém nový. Detailní popis nalezených komplikací s postupem jejich řešení je popsán v níže uvedených podkapitolách.

7.1 Programovací jazyk

Jako programovací jazyk pro náš projekt jsme si zvolili jazyk Python. Konkrétně Python ve verzi 2.7 a to díky velkému množství knihoven pro tuto verzi. Pro volbu tohoto programovacího jazyka nás vedl zejména fakt, že drtivou část času analýzy zabere práce s databází. Tato se nám potvrdila v budoucí části životního cyklu projekty, kdy nám práce s databází zabrala více jak 99,9 % času analýzy. Další výhodou jazyku Python je programovací komfort a díky syntaxi tohoto jazyka také rychlá orientace v kódu a to i v cizím.

7.2 SQL

V prvním prototypu jsme se rozhodli pro využití databáze SQL. Ta byla vybrána z prostého důvodu potřeby ukládání pouze hashe (který byl unikátním klíčem) a počtu jeho výskytů, tedy pouze vazba klíč - hodnota. Pro práci s databází bylo využito knihovny MySQLdb.

7.2.1 Nedostatečná rychlost ukládání

Prvním takto vytvořeným prototypem byla zjištěna nedostatečná rychlost ukládání dat do databáze. Nad databází SQL bylo potřeba při práci s každým vypočítaným hashem provést nejdříve operaci zjišťující, zda takovýto klíč v databázi již neexistuje. Až poté byl v databázi, na základě výsledku předešlé operace, tento klíč buď vytvořen a nebo u tohoto klíče byla inkrementována hodnota určující počet výskytů. U každého hashe bylo potřeba provést nad databází právě dvě operace. Je potřeba také zmínit, že při použití metody Rabin Fingerprint se v případě paketu velkého 1500 B vytvoří 1500 - β hashí. V případě použitý SQL databáze byla tedy časová náročnost ukládání neúnosně veliká a to i když byl testovací výpočetní stroj vybaven SSD diskem. Bylo tedy potřeba od využití této databáze upustit a nalézt alternativní řešení. Díky poznatkům získaným z vytvořeného prototypu

byla pro další prototyp vybrána jako vhodná databáze Redis. Co nás k volbě této databáze vedlo popíšeme v následující kapitole.

7.3 Redis

Databáze Redis se vyznačuje vysokou rychlostí zápisu, která je získána zejména prací na RAM pamětí výpočetního stroje. Rychlost této databáze můžeme pozorovat na benchmark testech A.2 respektive 7.2. První z testů je benchmark, který je přímo součástí instalace. Druhý je námi vytvořený benchmark, abychom mohli efektivně porovnávat s benchmarky ostatních databází (zdrojové kódy benchmark jsou k dispozici v příloze B). Kromě rychlosti navíc tato databáze disponuje některými dotazy, které svou činností nahradí více dotazů SQL databáze. Jako příklad takového dotazu je incr(). Tento dotaz provede buď inkrementování hodnoty nad příslušným id a nebo provede jeho vytvoření s hodnotou 1. Navíc je jeho časová složitost O(1) [7]. Pro náš prototyp se tedy jevil jako velice vhodný. Samozřejmě bylo potřeba počítat s omezenou kapacitou této databáze a to z již zmíněného důvodu prací nad RAM pamětí. V případě plného využitý RAM paměti začal systém očekávaně využívat odkládací prostor, tímto došlo ale k pochopitelnému zpomalení práci nad touto databází. Také bylo potřeba myslet i na budoucí použití a tak bylo ověřeno, že tato databáze zvládne i vhodné ukládání dat, které budeme potřebovat při budoucím vývoji aplikace.

Listing 7.1: Oficiální benchmark funkce incr() nad Redis

```
====== INCR ======
  10000 requests completed in 0.05 seconds
  50 parallel clients
  3 bytes payload
  keep alive: 1

100.00% <= 0 milliseconds
196078.44 requests per second
```

Listing 7.2: Náš benchmark funkce incr() pomocí jazyku Python

```
====== Redis ======
    10 000 requests completed in 0.37 seconds
====== Redis ======
    1 000 000 requests completed in 31.13 seconds
====== Redis ======
    10 000 000 requests completed in 340.66 seconds
```

7.3.1 Omezený prostor ukládání Redis

Již od počátku vytváření prototypu využívajícího Redis jsme si byli vědomi jeho omezené kapacity. Která byla zejména v případě využití metody Rabin Fingerprint pro analýzu objemných dat nedostačující. Proto jsme se v této fázi vývoje pokusili nalézt dostatečně rychlou databázi pracující nad pevným diskem. Z získaných informací jsme dospěli k databázi Berkeley DB. Zároveň jsme se ale vrátili opět do fáze návrhu metod s tím, že se pokusíme nalézt nebo navrhnout metodu produkující menší množství dat (oproti metodě Rabin Fingerprint) s co možná nejmenšími odchylkami ve výstupu aplikace. Díky tomuto prototypu jsme tedy modifikovali právě metodu Rabin Fingerprint. Tuto novou metodu jsme nazvali Rabin Fingerprint Offset Search (ta je již popsána v 4.5).

7.4 Berkeley DB

Berkeley DB je Open Source embedded databázový systém s řadou klíčových výhod oproti srovnatelným systémům. Pro nás je bezesporu největšími výhodami to, že je tato databáze schopná pracovat nad souborem uloženým přímo na disku a v kombinaci s SSD diskem nedojde k výraznému snížení rychlosti práce s takto uloženými daty. Fakt že jsme schopni pracovat nad databází uloženou přímo na disku řeší problém s nedostatkem paměti, který jsme řešili u Redis, kde jsme byly omezeni velikosti operační paměti. Tato vlastnost je samozřejmě vykoupena snížením výkonnosti databáze. Avšak při použití s již zmíněným SSD diskem je rychlost pro naší analýzu dostačující. I zde můžeme vidět výsledky našeho benchmark testu s velmi pozitivními výsledky. [3]

Listing 7.3: Náš benchmark použití Berkeley DB pomocí jazyku Python

```
====== Berkeley ======
    10 000 requests completed in 0.06 seconds
====== Berkeley ======
    1 000 000 requests completed in 7.37 seconds
====== Berkeley ======
    10 000 000 requests completed in 73.72 seconds
```

Kapitola 8

Výsledky testování

Na všemi výše popsanými vstupními daty z kapitoly 6 byla provedena analýza všemi implementovanými metodami. Výsledky jednotlivých analýz si blíže popíšeme v následujících podkapitolách

Parametr	Hodnota
CPU	Intel(R) Core(TM) i5-2500K CPU @ 3.30GHz
RAM	8079972 kB, 1867 MHz
HDD	KINGSTON SVP200S 90GB SSD
System	Linux version 3.8.0-19-generic (gcc version 4.7.3 (Ubuntu/-Linaro 4.7.3-1ubuntu1))

Tabulka 8.1: Konfigurace testujícího zařízení

Rabin Fingerprint
U metody Rabin Fingerprint jsme prováděli dvě sady testů s při různých nastavení parametrem β. Pochopitelně platí, že čím větší velikost β zvolíme, tím přísnější bude analýza při vyhledávání duplicity. Připomeňme, že velikost β udává jak velkou posloupnost bytů vyžadujeme pro detekování redundance.

Záznam	Redundance	Doba analýzy
google_png_2_times.pcap	50.08%	0m0.056s
google_png_10_times.pcap	89.69%	0m1.727s
idnes_2_times.pcap	49.10%	0m31.542s
music_from_file_hostings_3_times.pcap	65.69%	2m57.690s
youtube_2_times.pcap	47.71%	3m25.997s
big_one_file_no_redundancy.pcap	0.00%	66m28.755s

Tabulka 8.2: Rabin Fingerprint β = 32, M = pow(2, 60), p = 436507

Záznam	Redundance	Doba analýzy
google_png_2_times.pcap	50.08%	0m0.056s
google_png_10_times.pcap	89.69%	0m1.727s
idnes_2_times.pcap	47.40%	0m31.954s
music_from_file_hostings_3_times.pcap	64.88%	3m3.307s
youtube_2_times.pcap	47.18%	3m32.735s
big_one_file_no_redundancy.pcap	0.00%	58m57.467s

Tabulka 8.3: Rabin Fingerprint $\beta = 64$, M = pow(2, 60), p = 436507

Simhash

Z níže uvedených výsledků metody Simhash je patrné, že tato metoda v podrobených testech zcela neuspěla a to ať z hlediska dosažených výsledků, tak z hlediska rychlosti analýzy. Simhash je na detekování duplicity až příliš tolerantní a tak například u záznamu *big_one_file_no_redundancy.pcap* místo očekávaného výsledku 0% detekovala 59.96% duplicity. Tato metoda je tedy pro náš projekt absolutně nevhodná. Navíc použití Kosinové podobnosti při určování podobnosti jednotlivých paketů jsme dosahovali velkého počtu False Positive výsledků. Navíc z důvodu nepřiměřené doby analýzy jsme testování objemnějších záznamů již neprováděli a tuto metodu jsme prohlásili za nevhodnou pro analýzu reálného provozu.

Záznam	Redundance	Doba analýzy
google_png_2_times.pcap	54.27%	0m0.041s
google_png_10_times.pcap	90.94%	0m0.755s
idnes_2_times.pcap	74.51%	1m9.094s
music_from_file_hostings_3_times.pcap	59.78%	29m37.457s
youtube_2_times.pcap	70.42%	24m43.179s
big_one_file_no_redundancy.pcap	59.96%	434m37.851s

Tabulka 8.4: Simhash - Hammingova vzdálenost

Záznam	Redundance	Doba analýzy
google_png_2_times.pcap	55.26%	0m0.041s
google_png_10_times.pcap	91.49%	0m0.403s
idnes_2_times.pcap	99.68%	12m7.249s
music_from_file_hostings_3_times.pcap	testy neprováděny	
youtube_2_times.pcap		
big_one_file_no_redundancy.pcap		

Tabulka 8.5: Simhash - Kosinová podobnost

MD5

U výsledků analýzy prováděné touto metodou je nejzajímavější výsledek analýzy souboru *music_from_file_hostings_3_times.pcap*. U tohoto testu je výsledná redundance 0.00%. Tento

výsledek si vysvětlujeme posunutím přijímaných dat z různých serverů. Zejména tento výsledek nám ukazuje zásadní nedostatek této metody.

Záznam	Redundance	Doba analýzy
google_png_2_times.pcap	50.00%	0m0.030s
google_png_10_times.pcap	76.46%	0m0.046s
idnes_2_times.pcap	30.40%	0m0.109s
music_from_file_hostings_3_times.pcap	0.04%	0m0.435s
youtube_2_times.pcap	26.82%	0m0.399s
big_one_file_no_redundancy.pcap	0.00%	0m1.998s

Tabulka 8.6: MD5

MD5 token
Výsledky této metody jsou téměř totožné s metodou MD5, po které také „dědí“ její nedostatky. Výjimku tvoří pakety tvořené různými prohlížeči (přidáním různých metadat) aniž by došlo k posunu datové části paketu.

Záznam	Redundance	Doba analýzy
google_png_2_times.pcap	50.00%	0m0.031s
google_png_10_times.pcap	80.68%	0m0.076s
idnes_2_times.pcap	40.72%	0m0.855s
music_from_file_hostings_3_times.pcap	0.29%	0m7.172s
youtube_2_times.pcap	27.12%	0m5.945s
big_one_file_no_redundancy.pcap	0.00%	0m59.406s

Tabulka 8.7: MD5 token

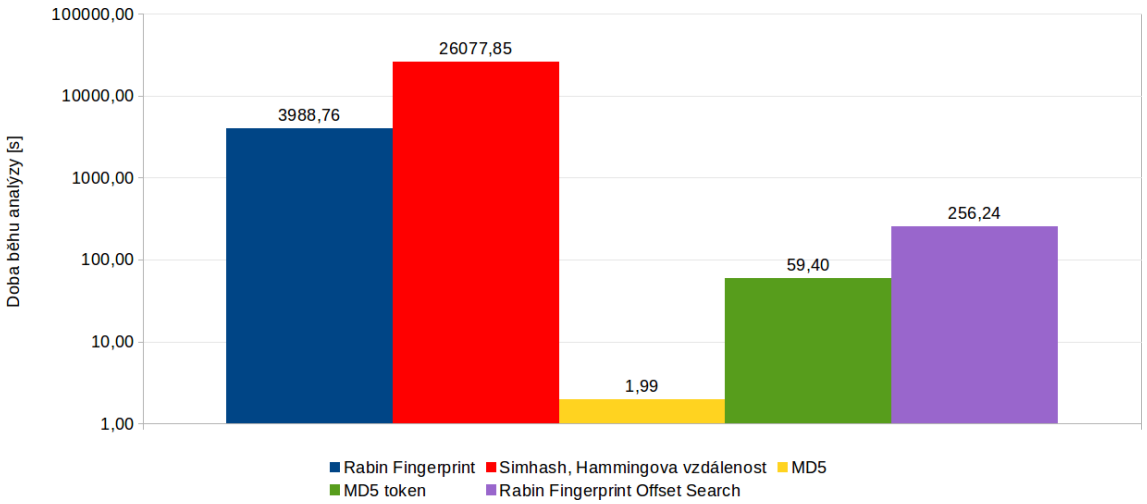
Rabin Fingerprint Offset Search
Výsledky testů prováděných touto metodou jsou velmi pozitivní. Nejenže se blížíme očekávaným výsledkům, ale také rychlost analýzy je uspokojující. Jediným méně přesným výsledkem je analýza souboru *music_from_file_hostings_3_times.pcap* u kterého jsme se odchýlili od reálného výsledku o cca 20%.

Záznam	Redundance	Doba analýzy
google_png_2_times.pcap	50.00%	0m0.038s
google_png_10_times.pcap	87.67%	0m0.434s
idnes_2_times.pcap	47.44%	0m6.269s
music_from_file_hostings_3_times.pcap	42.61%	0m33.333s
youtube_2_times.pcap	40.67%	0m37.280s
big_one_file_no_redundancy.pcap	0.00%	4m16.241s

Tabulka 8.8: Rabin Fingerprint Offset Search

8.1 Rychlost analýzy

Pokud jednotlivé metody seřadíme dle rychlosti jejich analýzy od nejrychlejší po nejpomalejší, pak dostaneme následující posloupnost MD5, MD5 token, Rabin Fingerprint Offset Search, Rabin Fingerprint, Simhash. Ne vždy však pomalejší analýza vedla k lepším výsledkům. Ukázkovým důkazem tohoto tvrzení je metoda Simhash. Rozdíl v rychlostech analýzy můžeme vidět na grafu 8.1, který znázorňuje dobu běhu programu při zpracovávání souboru *big_one_file_no_redundancy.pcap*

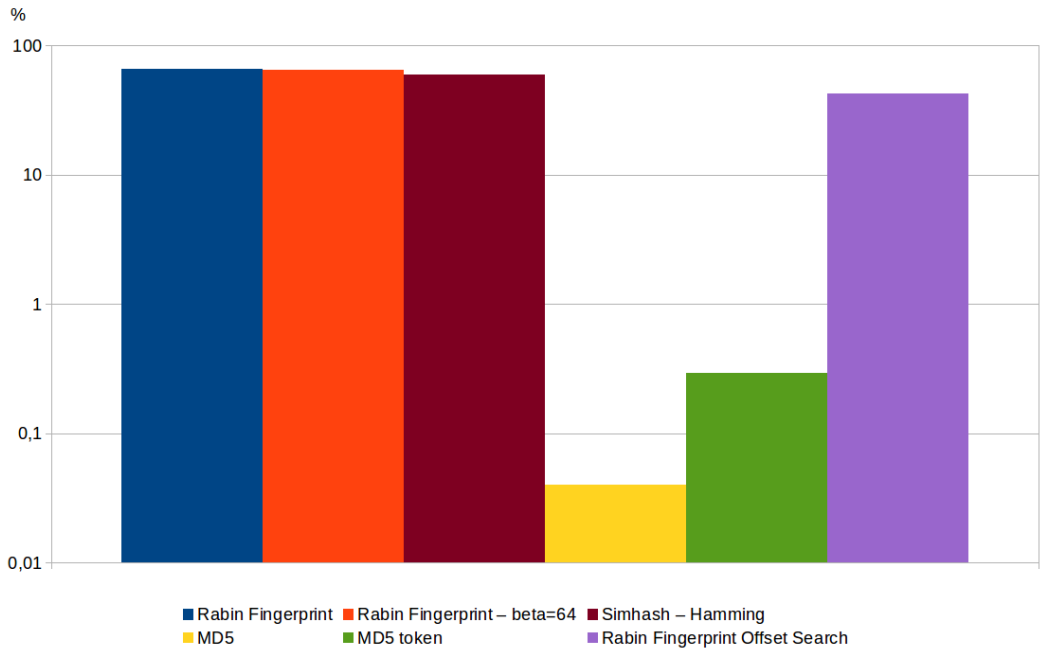


Obrázek 8.1: Doba analýzy souboru big_one_file_no_redundancy.pcap

8.2 Zhodnocení testů

Pokud se na získané výsledky podíváme jako na celek. Zjistíme, že nejpřesnějšími metodami jsou metody Rabin Fingerprint a její modifikace Rabin Fingerprint Offset Search. Je však také vidět, že tato přesnost je u těchto metod vykoupena rychlostí analýzy (máme na mysli především u metody Rabin Fingerprint), jejíž rozdíl je pozorovatelný již u analýzy takovýchto „malých“ záznamů. U analýzy obsáhlejších záznamů by byl tento rozdíl ještě výraznější a rostl by exponenciálně. Z tohoto důvodu a díky velké paměťové náročnosti bude metoda Rabin Fingerprint vhodná pro analýzy nevelkých záznamů, kde budeme chtít dosáhnout co největší přesnosti.
Metody využívající generování hashe/hashí na základě algoritmu MD5 nedosahují špatných výsledků, ale k jejich velkým výhodám patří jejich rychlost a jednoduchost. Tyto metody budou vhodné pro analýzy opravdu velkých záznamů.
Dále je potřeba zdůraznit výsledky metody Simhash. Tyto výsledky nám ukázaly velké nedostatky pro použití v našem projektu (rychlost, nepřesnost). Z těchto důvodu nebudeme touto metodou reálný provoz analyzovat.
Velmi důležitým faktem tohoto testování je zajisté také, že kromě metody Simhash nedochází u ostatních metod k False Positive detekcím a tak získané výsledky můžeme považovat za dolní hranici zastoupení duplicitního provozu v analyzovaném souboru.
Pro přehlednost ještě graficky porovnáme výsledky jednotlivých metod nad testem, který

podle nás nejvíce prověřil kvality jednotlivých metod. Jedná se o test nad záznamem *music_from_file_hostings_3_times.pcap* (viz graf 8.2). V tomto testu je zejména zřetelný neúspěch detekce pomocí metod MD5 a MD5 token, které z důvodu různého rozdělení datového streamu do paketů nebyly schopni redundanci detekovat. Z téhož důvodu došlo u metody Rabin Fingerprint Offset Search u tohoto testu k největší odchylce od očekávaného výsledku. Stejně tak jako u ostatních testů nejlépe uspěla metoda Rabin Fingerprint.
Díky těmto testům můžeme (na základě našich požadavků přesnosti a velikosti vstupního záznamu) velmi dobře zvolit tu nejvhodnější metodu detekování duplicitního provozu.



Obrázek 8.2: Výsledky metod nad záznamem music_from_file_hostings_3_times.pcap

Kapitola 9

Výsledky analýzy reálného provozu

9.1 Očekávané výsledky

Očekávané výsledky se u tohoto tématu opravdu těžko odhadují. V první řadě je potřeba vzít v potaz nad jakými daty budeme detekci provádět. Záznam provozu bude pořízen z páteřního zařízení sítě VUT, konkrétně se bude jednat o páteřní síť kolejní sítě KolejNet. Tato síť se nachází v jedné lokalitě (tedy v jednom městě) a její uživatelé jsou si navíc profesně velmi blízcí. Proto se dá očekávat zájem o podobné (ne-li totožné) služby na internetu. Jako další významný faktor ovlivňující výsledek je doba po kterou bude záznam pořízen (respektive množství pořízených dat). Je potřeba si uvědomit, že z počátku záznamu nebudeme mít redundanci s čím porovnávat a tak na začátku záznamu bude docházet pouze k opravdu minimální redundanci. Pokud budeme mít dostatečnou velikost pořízeného záznamu, tak si můžeme dovolit odhadnout procentuální zastoupení redundantního provozu někde okolo 10%. Jedním z důvodů (kromě již zmíněných) pro toto číslo také zajisté je, že s neustálým nárůstem rychlosti připojení koncových uživatelů roste i velikost přenášených dat (populárnější weby jsou oproti ostatním často graficky náročnější, kvalitnější multimedia atd.). Stále se ale jedná pouze o odhadnutou cifru, od které se v praxi můžeme značně odchýlit. Tyto odhady mohou být navíc ovlivněny pokročilými cashovacími systémy jednotlivých prohlížečů. Proto je potřeba na tyto odhady nahlížet s dostatečnou rezervou. Proto abychom mohli získané výsledky prohlásit za co nejpřesnější by nám měly posloužit testy nad analyzátorem a jeho jednotlivými metodami detekce. Návrhy testů pro tuto aplikaci si probereme v následující kapitole.

9.2 Statistiky provozu

Provoz, který jsme podrobili analýze, pocházel z páteřní sítě VUT. Konkrétně se jednalo o síť KolejNet. Z tohoto provozu bylo vybráno, podle dlouhodobých statistik, deset nejaktivnějších uživatelů. Provoz těchto uživatelů byl zaznamenáván po dobu 24 hodin. Za tuto dobu bylo těmito uživateli přeneseno 38,64 GB (konkrétně 41 487 457 561 B) užitečných dat (tedy bez hlaviček potřebných pro režii komunikace, konkrétně hlaviček 4., 3. a 2. vrstvy ISO/OSI modelu). Tento provoz byl rozdělen na 45 souborů a velikosti cca 1GB. Díky tomu bylo jednodušší zaznamenávání výsledků detekce po jednotlivých krocích, kterou můžeme vidět na jednotlivých grafech níže. Z celkové analýzy provozu vycházejí zajímavá data. A to například fakt, že z top 25 navštěvovaných serverů je celých 13 provozujících službu sdílení dat www.uloz.to společnosti Nodus Technologies spol. s.r.o. Využíváním této služby bylo

ze zachyceného provozu přeneseno 15,16 GB (přesně 16 280 218 916 B) tedy 39,24% celkového provozu (viz tabulka 9.1). Přehledně jsou tyto statistiky zobrazeny v následujících tabulkách a grafech.

Server/služba (počet IP adres)	**Zastoupení v provozu [%]**
uloz.to (13)	39,24
visuality (1)	5,94
Mafra (1)	3,36
Google (4)	3,28
ROSTELECOM (1)	1,81
EdgeCast Networks (1)	1,02
Netlook (1)	0,76
GTS Slovakia (1)	0,66
pipni.cz (1)	0,53
TVK-AS-TECHNET3 (1)	0,53
zbylé	42,86

Tabulka 9.1: Rozložení serverů/služeb analyzovaného toku

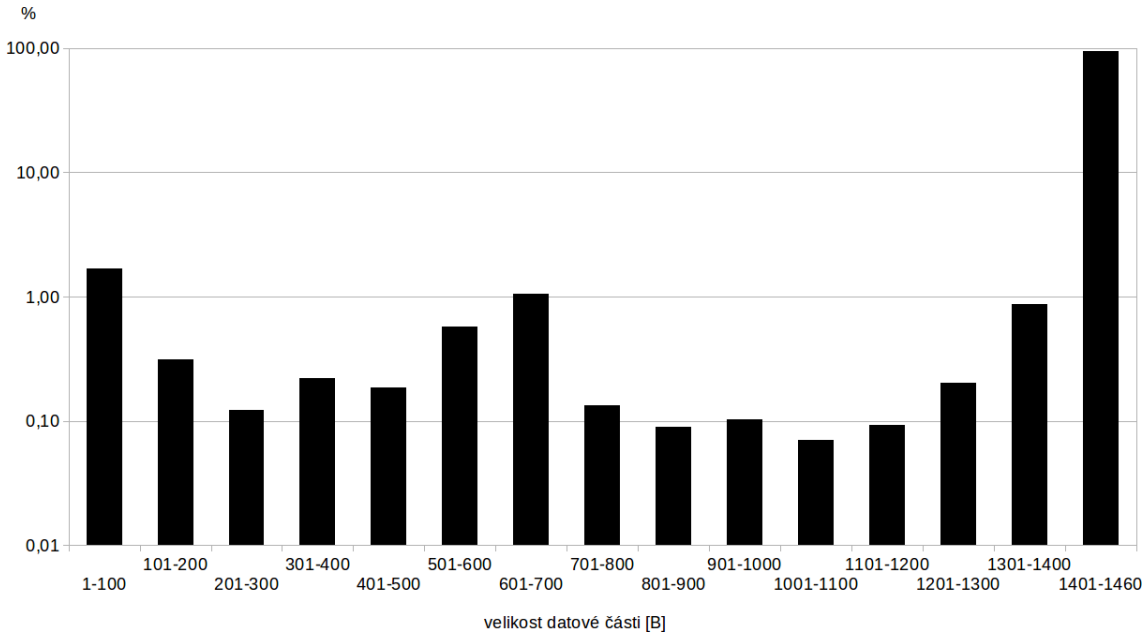
Velikost dat	**Procentuální zastoupení**
1 - 100	1,68
101 - 200	0,31
201 - 300	0,12
301 - 400	0,22
401 - 500	0,19
501 - 600	0,58
601 - 700	1,06
701 - 800	0,13
801 - 900	0,09
901 - 1000	0,10
1001 - 1100	0,07
1101 - 1200	0,09
1201 - 1300	0,20
1301 - 1400	0,87
1401 - 1460	94,29

Tabulka 9.2: Tabulka rozložení velikosti přenášených dat

Z grafů je dále patrné, že těchto 25 top serverů, které byly navštěvovány, nám „zabírá“ více jak 57% datového provozu. Předpokládáme ovšem, že takto velkého procentuálního zastoupení je dosaženo díky volbě nejaktivnějších uživatelů. V případě analýzy většího počtu uživatelů by došlo k většímu rozprostření spektra. Dále je tato situace ovlivněna analýzou pouze nešifrovaného provozu. Pokud by byla analýza prováděna i nad šifrovaným provozem, tak by se zajisté mezi top servery objevila například také služba Facebook.

Dalším zajímavým faktorem síťového provozu je rozložení podle velikosti přenášených

dat (viz. tabulka 9.2). Opět máme na mysli užitečná data. Toto rozložení je znázorněno na grafu 9.1. Jak je na první pohled vidět, tak v drtivé většině síťového provozu se jednalo o pakety omezené nastaveným MTU (Maximum transmission unit). Průměrná velikost přenesených dat na jeden paket byla v tomto provozu 702,37 B (počítáno z paketů s alespoň jedním bytem užitečných dat).



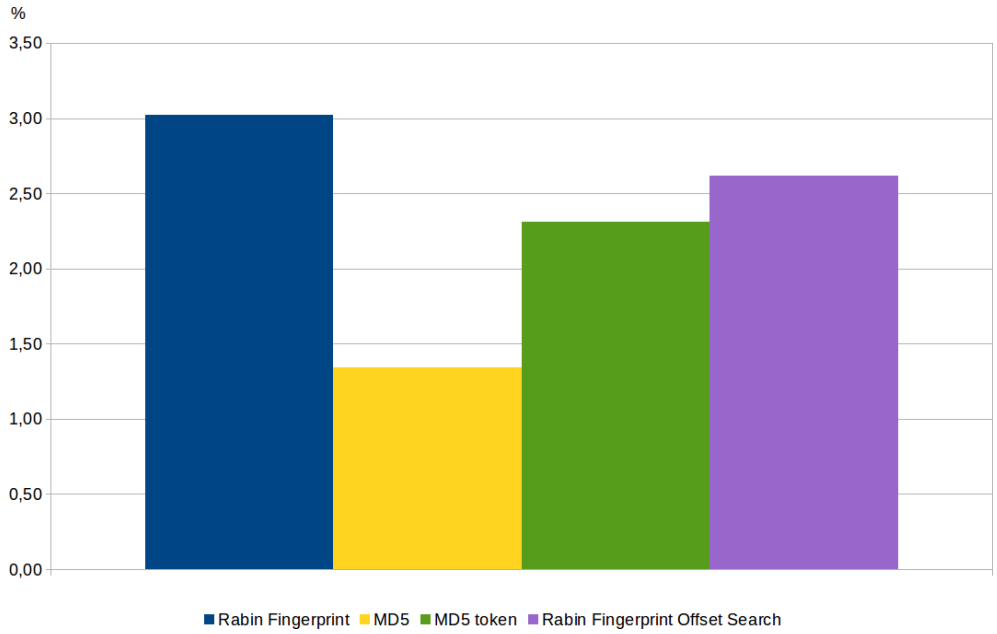
Obrázek 9.1: Graf rozložení velikosti přenášených dat

9.3 Výsledky duplicitní analýzy

V následujících podkapitolách si postupně rozebereme výsledky analýzy získaného provozu za použití jednotlivých metod. V těchto kapitolách se zaměříme zejména na rozdíly ve výsledcích jež tyto metody produkují.

9.3.1 Metoda Rabin Fingerprint

Analýza touto metodou byla časově i paměťově velice náročná (z důvodů zmíněných v kapitole 4.1) avšak analýza touto metodou je velice přesná a tak dosažený výsledek můžeme ze všech ostatních považovat za nejpřesnější. Díky velké časové a zejména paměťové náročnosti bylo možno touto metodou provést analýzu pouze na relativně malou část provozu (konkrétně na jeden ze souborů, tedy na cca 1GB provoz). Analýza celého zaznamenaného provozu by i za podmínky, že bychom vlastnili SSD disk o kapacitě cca 1 TB, trvala dle odhadů přibližně 30 dní. Z analyzovaného souboru byla zjištěna 3,02% duplicitního provozu (konkrétně se jedná o 28029902 B z celkových 928142378 B). Porovnání s výsledky ostatních metod u téhož souboru můžeme vidět na grafu 9.2.



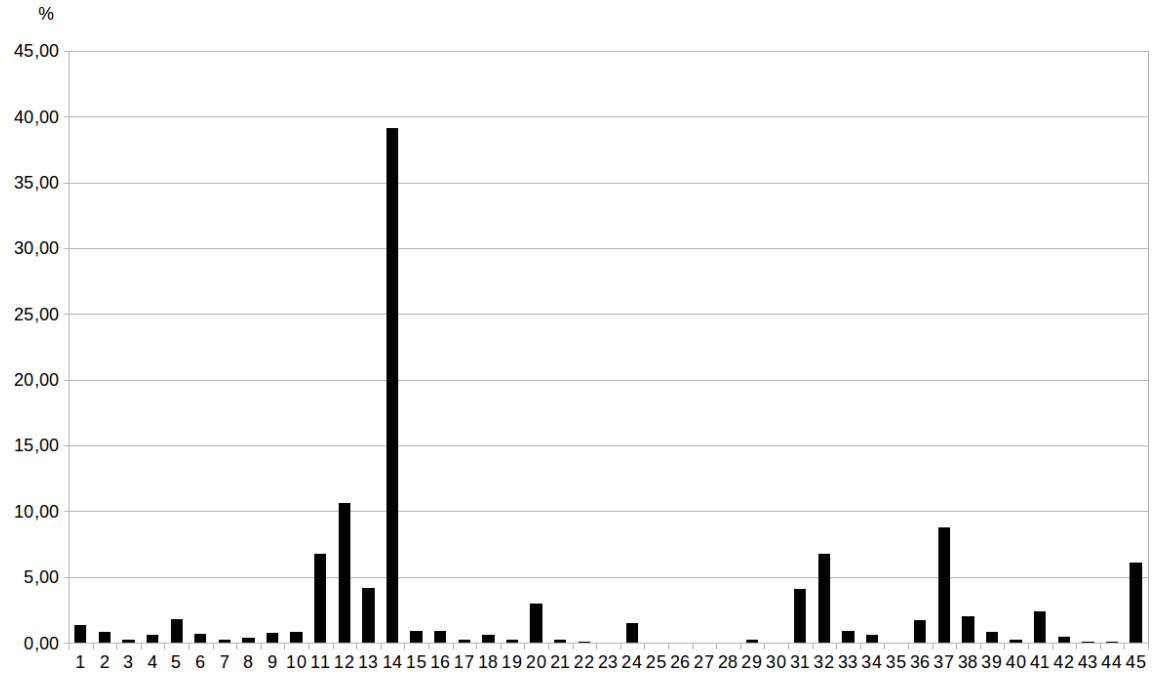
Obrázek 9.2: Porovnání výsledků analýzy nad jedním záznamem o velikosti 1 GB

9.3.2 Metoda MD5

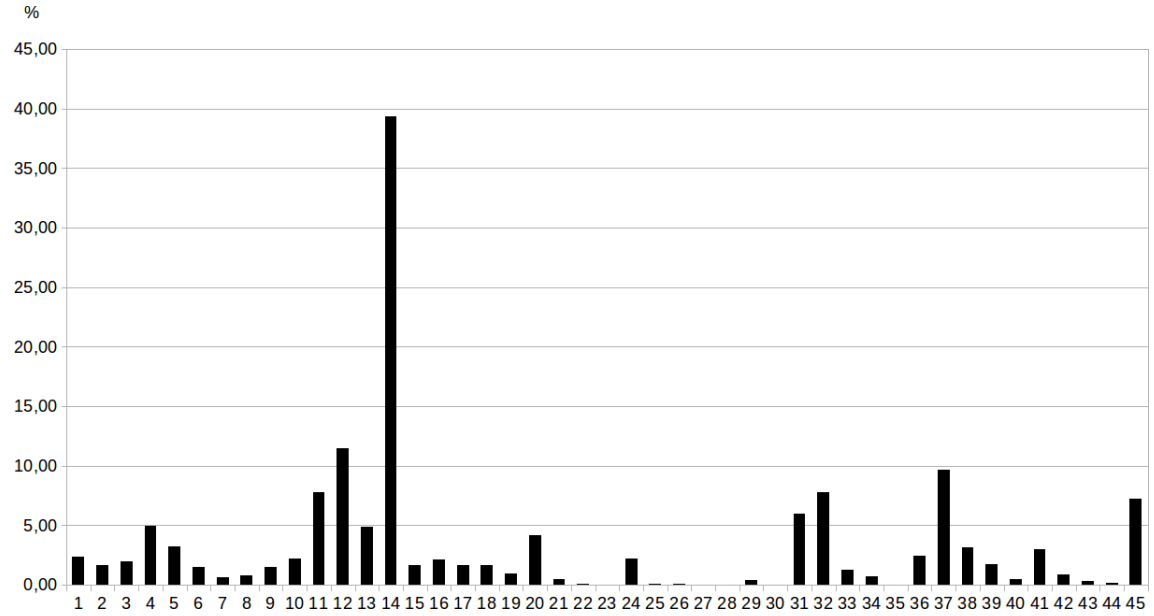
Analýzou záznamu za použitím této metody jsme dospěli k následujícím závěrům. Nejdůležitější je samozřejmě celková redundance. Ta u analýzy touto metodou byla 2,48%. Ač se toto číslo zdá velmi malé, tak je třeba poznamenat, že v tomto případě se jednalo o zcela totožné pakety. Pokud toto číslo přepočítáme na celkový objem duplicitních dat, tak se dostaneme k cifře 0,96 GB (přesněji 1 028 888 948 B). Následující graf zobrazuje průběh výpočtu po jednotlivých sekcích. Na grafu 9.3 můžeme vidět průběh analýzy po jednotlivých záznamech. Tedy procentuální duplicitu jednotlivých bloků (vzhledem i k minulým sekcím). Již na první pohled na grafu zaujme určitá anomálie ve 14. bloku analýzy. Tuto anomálii se pokusíme zdůvodnit v jedné z následujících kapitol (viz 9.4).

9.3.3 Metoda MD5 token

Metodou MD5 token jsme analýzou dospěli k výsledku, že získaný záznam obsahuje 3,20% redundantního provozu. Což činí 1,24 GB (přesně 1 328 469 879 B) z celkového provozu. Touto metodou tedy došlo oproti metodě MD5 k nárůstu detekování redundantního provozu o 29,12%. Tedy u necelé třetiny provozu dochází ke změně pouze u části paketu. Tento fakt je v důsledku například měnících se stahovaných prvků (jako například čas u navštívených stránek) nebo z důvodu použití jiných internetových prohlížečů. Průběh detekce touto metodou můžeme vidět na grafu 9.4.



Obrázek 9.3: Graf průběhu výpočtu redundance metody MD5

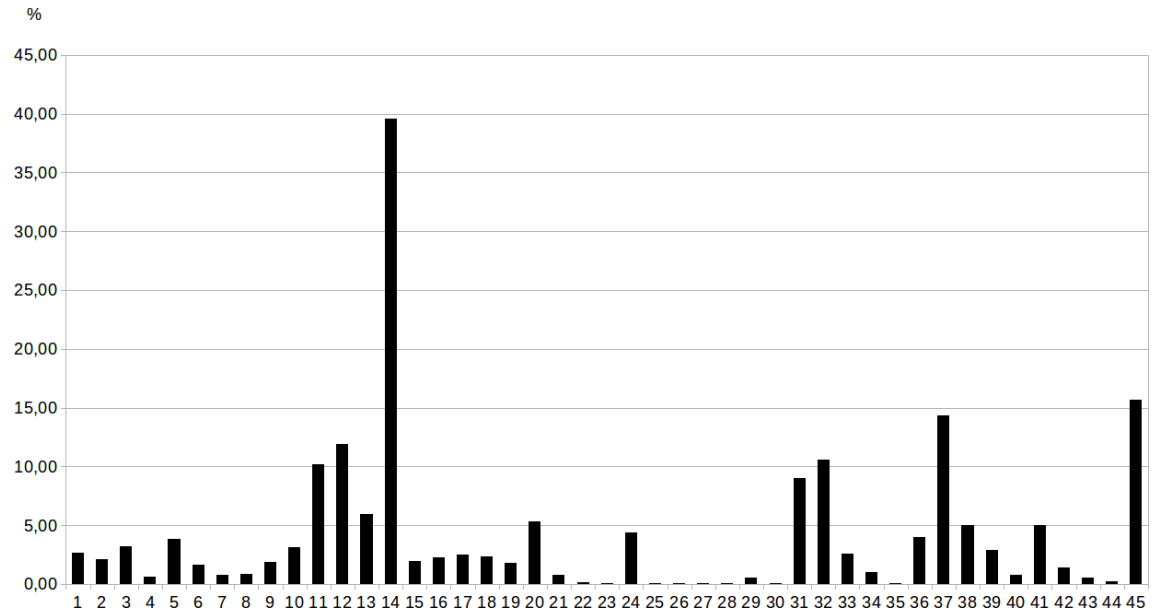


Obrázek 9.4: Graf průběhu výpočtu redundance metody MD5 token

9.3.4 Metoda Rabin Fingerprint Offset Search

Díky proběhlým obsáhlým testů v kapitole 8 můžeme s jistotou tvrdit, že výsledky získané touto metodou jsou nejpřesnější (samozřejmě pomineme-li výsledek metodou Rabin Fingerprint, která analyzovala pouze část provozu) které můžeme za daných podmínek dosáhnout a proto se jejich výstupy budeme zabývat o něco podrobněji než předešlým metodám. Výsle-

dek analýzy touto metodou je 4,07%. Pro upřesnění opět uvedeme velikost redundantního provozu v bytech. Ta činí 1,57 GB (přesně se jedná o 1 689 636 603 B). U této metody došlo tedy oproti metodě MD5 token k nárůstu o 27% a oproti metodě MD5 dokonce o 64%. Průběh analýzy touto metodou můžeme, stejně jako u předešlých metod, vidět na grafu 9.5. V tomto grafu je vidět postupná tendence zvyšování detekovaného duplicitního provozu. Věříme, že v případě ještě většího analyzovaného provozu by k tomuto nárůstu ještě docházelo a my bychom se mohli našimi výsledky přiblížit k odhadovaným 10% redundantního provozu.



Obrázek 9.5: Graf průběhu výpočtu redundance metoda Rabin Fingerprint Offset Search

V následující tabulce 9.3 můžeme vidět procentuální zastoupení jednotlivých serverů/služeb, které se podíleli na duplicitě. V této tabulce je potřeba si povšimnout služby uloz.to, která ač v celkovém síťovém provozu patřila s 39,24% k jednoznačné top službě, tak její celkový podíl na duplicitním provozu činil pouze 0,82%. Dále je na první pohled zřejmé, že duplicitní provoz je rozprostřen mezi servery rovnoměrněji než provoz celkový. V neposlední řadě je nutné také zmínit velký podíl duplicitního provozu společnosti Google. V tomto ohledu je služba youtube.com k jednoznačným top službám generujícím duplicitní provoz.

Server/služba (počet IP adres)	**Zastoupení v provozu [%]**
Google (4)	10,93
visuality (1)	8,31
superhosting.cz (2)	1,80
vhosting.cz (1)	1,34
Mafra (1)	1,18
uloz.to (3)	0,82
zbylé	75,61

Tabulka 9.3: Tabulka serverů/služeb, kteří se podíleli na duplicitě (počet IP adres)

Získanými výsledky jsme se sice nepřiblížili odhadnutým 10%, ale věříme že v případě záznamu, který by zaznamenal větší časový úsek, bychom byli schopni zmíněných 10% detekovat.

9.4 Anomálie analyzovaného provozu

Z grafů jednotlivých průběhů analýzy vyčnívá výsledek detekce 14. bloku (např. viz graf 9.3). Při bližší analýze tohoto provozu jsme zjistili, že se v tomto bloku nachází provoz přenášející velké množství dat z databáze Microsoft SQL Serveru. Komunikace vedoucí z tohoto serveru tvoří 57,01% záznamu tohoto bloku. Podobná komunikace pobíhala již o jeden blok dříve. Patrně se tedy jednalo o opakované zálohování této databáze.

Kapitola 10

Závěr

Zadáním diplomové práce nám bylo uloženo seznámit se v odborné literatuře s dostupnými technikami detekce duplicitního provozu v síti a následně na základě získaných informací o dostupných technikách navrhnout nástroj, který bude duplicitní síťový provoz detekovat. Tento nástroj následně implementovat a podrobit testům nad záznamy síťového provozu.

Z odborného článku „A protocol-independent technique for eliminating redundant network traffic“ [9] ze kterého je v této práci také citováno byla nalezena metoda Rabin Fingerprint. Tato metoda byla doplněna metodou vhodnou pro detekci podobnosti Simhash [8]. Společně s touto metodou bylo navrženo několik dalších metod. Konkrétně se jedná o metody MD5, MD5 token a modifikaci metody Rebin Fingerprint, která byla nazvána Rabin Fingerprint Offset Search. Všechny tyto metody byly v následujícím životním cyklu projektu implementovány podle výše popsaného návrhu. Fakt, že jsme se rozhodli implementovat všechny tyto metody, nám umožnil velmi efektivně otestovat jejich kvality. Navíc je potřeba zdůraznit, že u většiny metod (výjimku tvoří metoda Simhash) je přímá úměra mezi časovou/paměťovou náročností a kvalitou dosažených výsledků. Uživatel implementovaného analyzátoru si tedy může vhodně volit dle přesnosti dosažených výsledků.

Dále je potřeba zdůraznit, že námi navržená metoda Rabin Fingerprint Offset Search během testování dokázala své kvality a při analyzování reálného provozu se stala nejpřesnější metodou, kterou bylo možno na takto velký objem dat použít.

Věřím, že získané výsledky a aplikace najdou uplatnění při detekci duplicitního provozu a že tato práce vytváří základ pro možnost dalšího budoucího vývoje tohoto tématu. Možnosti budoucího rozšíření aplikace si popíšeme v jedné z následujících podkapitol.

10.1 Omezení analyzátoru

Za omezení aplikace je potřeba označit možnosti detekce „pouze“ nešifrovaného provozu. Dále je třeba vzít v potaz a také nutné označit jako nedostatek rychlost analýzy v případě použití určitých metod detekce. Zejména máme na mysli metodu Rabin Fingerprint, která je bezesporu na analýzu nejnáročnější. Pokud se však bavíme o náročnosti analýzy, pak se ve velké části případů bavíme o náročnosti ukládání a celkové práce s daty respektive s databází.

10.2 Možnosti rozšíření aplikace

Budoucí vývoj této aplikace by se mohl zaměřit například na sestavení možných streamových proudů z záznamu síťové komunikace. V takovémto případě bychom mohli počítat i s tokeny které jsou rozděleny do různých paketů. Jako další možností vývoje se nabízí samozřejmě detekce šifrovaného provozu.

Literatura

[1] Bayardo, R. J.; Ma, Y.; Srikant, R.: Scaling up all pairs similarity search. In *Proceedings of the 16th international conference on World Wide Web*, WWW '07, New York, NY, USA: ACM, 2007, ISBN 978-1-59593-654-7, s. 131–140, doi:10.1145/1242572.1242591.
URL http://doi.acm.org/10.1145/1242572.1242591

[2] Gupta, A.; Akella, A.; Seshan, S.; aj.: Understanding and Exploiting Network Traffic Redundancy. 2007.

[3] Olson, M. A.; Bostic, K.; Seltzer, M.: Berkeley DB. 1999.

[4] Postel, J.: User Datagram Protocol. RFC 768, Internet Engineering Task Force, August 1980.
URL http://www.rfc-editor.org/rfc/rfc768.txt

[5] Postel, J.: RFC 791 Internet Protocol - DARPA Inernet Programm, Protocol Specification. Technická zpráva, Internet Engineering Task Force, 1981.
URL http://www.rfc-editor.org/rfc/rfc791.txt

[6] Postel, J.: Transmission Control Protocol. RFC 793, Internet Engineering Task Force, September 1981.
URL http://www.rfc-editor.org/rfc/rfc793.txt

[7] Redis, C.: INCR key. [online], 4. 5. 2013.
URL http://redis.io/commands/incr

[8] Sood, S.; Loguinov, D.: Probabilistic near-duplicate detection using simhash. In *Proceedings of the 20th ACM international conference on Information and knowledge management*, CIKM '11, New York, NY, USA: ACM, 2011, ISBN 978-1-4503-0717-8, s. 1117–1126, doi:10.1145/2063576.2063737.
URL http://doi.acm.org/10.1145/2063576.2063737

[9] Spring, N. T.; Wetherall, D.: A protocol-independent technique for eliminating redundant network traffic. *SIGCOMM Comput. Commun. Rev.*, ročník 30, č. 4, Srpen 2000: s. 87–95, ISSN 0146-4833, doi:10.1145/347057.347408.
URL http://doi.acm.org/10.1145/347057.347408

[10] Tineye: Results. [online], 4. 1. 2013.
URL http://tineye.com/search/3ee0d360dc12003c0d43e3579295b52b64906e85/

[11] Youtube: Statistiky. [online], 5. 1. 2013.
URL http://www.youtube.com/t/press_statistics

Dodatek A

Uživatelská příručka

A.1 Instalace

Pro běh programu je vyžadována instalace Python2.7, Redis, python-pcapy

Příklad instalace za použitím apt-get:

```
sudo apt-get install python

sudo apt-get install redis-server

sudo apt-get install python-pcapy
```

A.2 Ovládání programu

```
POPIS
       program pro detekci duplicitniho provozu z *.pcap souboru. Konfiguraci
           programu muzete upravit v souboru config.py

FORMAT
       analyze.py [OPTION] FILE

OPTION
       Program ocekava pouze jediny argument a to soubor z ktereho se bude analyza
           provadet. Ostatni parametry jsou volitelne.

       Neni mozne provadet vypocet dvema metodami najednou.

       Pouzivane metody jsou blize popsany v textove casti diplomove prace.

       Pokud neni zvoleno jinak, je defaultni metodou pro vypocet duplicity metoda
           MD5 tokenu a databaze Berkeley DB.

       -d
              pred startem analyzy vymaze databazi

       -m
              metoda MD5 token

       -M
              metoda MD5

       -r
              metoda Rabin Fingerprint

       -s
```

```
              metoda Simhash s detekci Kosinove podobnosti

       -S
              metoda Simhash s detekci Hammingovy vzdalenosti

       -o
              metoda Rabin Fingerprint Offset Search

       -R
              vyuzije Redis (neni mozne kombinovat s -B, -s, -S)

       -B
              vyuzije Berkeley DB (neni mozne kombinovat s -R).

       -b
              debug mode

PRIKLAD
       analyze.py -d -o input.pcap      Pouzije databazi Berkeley DB, tu vymaze a
           metodou Rabin Fingerpint zanalyzuje soubor input.pcap
```

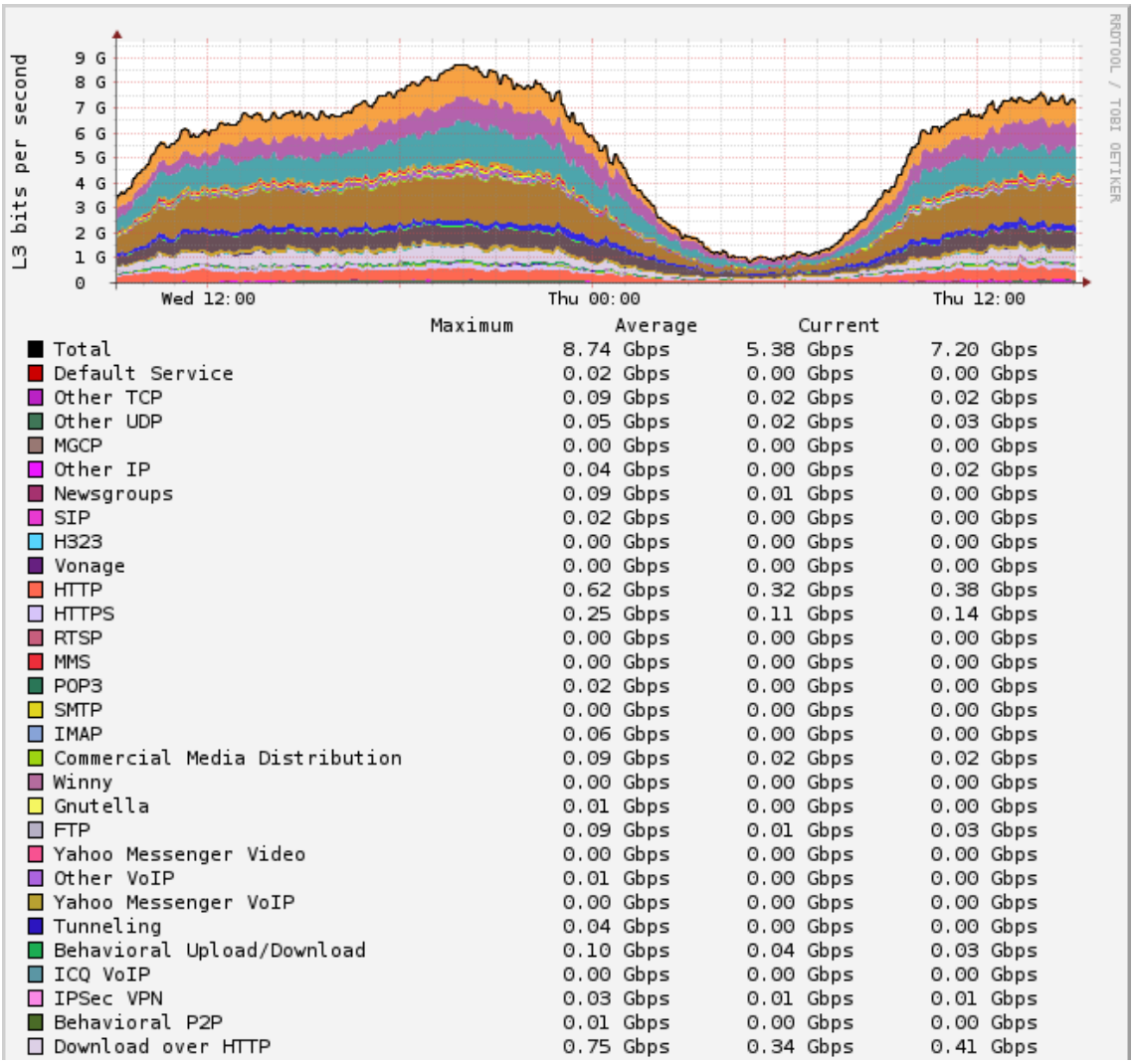
Dodatek B

Obsah DVD

- src - zdrojové kódy
  - benchmarks - zdrojové kódy benchmarkových testů
- test_pcap - seznam záznamů síťového provozu nad kterými byly prováděny testy
- outputs - výstupy analýz jednotlivých metod a jednotlivých testů
- README - informace o projektu
- xkrchp00.pdf - tato diplomová práce v elektronické podobě

Dodatek C

Kompletní graf přenášených služeb z Service Control Engine 8000



	Maximum	Average	Current
Total	8.74 Gbps	5.38 Gbps	7.20 Gbps
Default Service	0.02 Gbps	0.00 Gbps	0.00 Gbps
Other TCP	0.09 Gbps	0.02 Gbps	0.02 Gbps
Other UDP	0.05 Gbps	0.02 Gbps	0.03 Gbps
MGCP	0.00 Gbps	0.00 Gbps	0.00 Gbps
Other IP	0.04 Gbps	0.00 Gbps	0.02 Gbps
Newsgroups	0.09 Gbps	0.01 Gbps	0.00 Gbps
SIP	0.02 Gbps	0.00 Gbps	0.00 Gbps
H323	0.00 Gbps	0.00 Gbps	0.00 Gbps
Vonage	0.00 Gbps	0.00 Gbps	0.00 Gbps
HTTP	0.62 Gbps	0.32 Gbps	0.38 Gbps
HTTPS	0.25 Gbps	0.11 Gbps	0.14 Gbps
RTSP	0.00 Gbps	0.00 Gbps	0.00 Gbps
MMS	0.00 Gbps	0.00 Gbps	0.00 Gbps
POP3	0.02 Gbps	0.00 Gbps	0.00 Gbps
SMTP	0.00 Gbps	0.00 Gbps	0.00 Gbps
IMAP	0.06 Gbps	0.00 Gbps	0.00 Gbps
Commercial Media Distribution	0.09 Gbps	0.02 Gbps	0.02 Gbps
Winny	0.00 Gbps	0.00 Gbps	0.00 Gbps
Gnutella	0.01 Gbps	0.00 Gbps	0.00 Gbps
FTP	0.09 Gbps	0.01 Gbps	0.03 Gbps
Yahoo Messenger Video	0.00 Gbps	0.00 Gbps	0.00 Gbps
Other VoIP	0.01 Gbps	0.00 Gbps	0.00 Gbps
Yahoo Messenger VoIP	0.00 Gbps	0.00 Gbps	0.00 Gbps
Tunneling	0.04 Gbps	0.00 Gbps	0.00 Gbps
Behavioral Upload/Download	0.10 Gbps	0.04 Gbps	0.03 Gbps
ICQ VoIP	0.00 Gbps	0.00 Gbps	0.00 Gbps
IPSec VPN	0.03 Gbps	0.01 Gbps	0.01 Gbps
Behavioral P2P	0.01 Gbps	0.00 Gbps	0.00 Gbps
Download over HTTP	0.75 Gbps	0.34 Gbps	0.41 Gbps

Windows Live Messenger VoIP	0.00 Gbps	0.00 Gbps	0.00 Gbps
MS Push Mail	0.00 Gbps	0.00 Gbps	0.00 Gbps
Location Based Services	0.01 Gbps	0.00 Gbps	0.00 Gbps
One-Click Hosting	0.02 Gbps	0.01 Gbps	0.01 Gbps
QQ File Transfer	0.00 Gbps	0.00 Gbps	0.00 Gbps
Yahoo Messenger File Transfer	0.00 Gbps	0.00 Gbps	0.00 Gbps
Google Talk File Transfer	0.00 Gbps	0.00 Gbps	0.00 Gbps
ICQ File Transfer	0.00 Gbps	0.00 Gbps	0.00 Gbps
Other IM File Transfer	0.00 Gbps	0.00 Gbps	0.00 Gbps
Windows Live Messenger File Transfer	0.00 Gbps	0.00 Gbps	0.00 Gbps
Ares/Warez	0.00 Gbps	0.00 Gbps	0.00 Gbps
Other P2P	0.03 Gbps	0.00 Gbps	0.01 Gbps
Encrypted eMule	0.01 Gbps	0.00 Gbps	0.00 Gbps
Non-Encrypted eMule	0.01 Gbps	0.00 Gbps	0.00 Gbps
Encrypted Bittorrent	0.17 Gbps	0.10 Gbps	0.11 Gbps
Non-Encrypted Bittorrent	0.72 Gbps	0.50 Gbps	0.61 Gbps
Behavioral VoIP	0.00 Gbps	0.00 Gbps	0.00 Gbps
SkypeIn	0.00 Gbps	0.00 Gbps	0.00 Gbps
SkypeOut	0.00 Gbps	0.00 Gbps	0.00 Gbps
Other Skype	0.06 Gbps	0.02 Gbps	0.02 Gbps
Google Talk VoIP	0.00 Gbps	0.00 Gbps	0.00 Gbps
QQ VoIP	0.00 Gbps	0.00 Gbps	0.00 Gbps
Other Flash	0.30 Gbps	0.18 Gbps	0.27 Gbps
Flash MySpace	0.00 Gbps	0.00 Gbps	0.00 Gbps
Flash YouTube	0.00 Gbps	0.00 Gbps	0.00 Gbps
Flash Yahoo	0.00 Gbps	0.00 Gbps	0.00 Gbps
Audio and Video over HTTP	1.84 Gbps	1.08 Gbps	1.63 Gbps
Other P2P TV	0.10 Gbps	0.02 Gbps	0.02 Gbps
PPLive	0.00 Gbps	0.00 Gbps	0.00 Gbps
PPStream	0.00 Gbps	0.00 Gbps	0.00 Gbps
Joost	0.00 Gbps	0.00 Gbps	0.00 Gbps
Windows Live Messenger	0.00 Gbps	0.00 Gbps	0.00 Gbps
Google Talk	0.00 Gbps	0.00 Gbps	0.00 Gbps
Yahoo Messenger	0.00 Gbps	0.00 Gbps	0.00 Gbps
ICQ	0.00 Gbps	0.00 Gbps	0.00 Gbps
Other Instant Messaging	0.00 Gbps	0.00 Gbps	0.00 Gbps
PC Gaming	0.12 Gbps	0.03 Gbps	0.01 Gbps
Xbox	0.00 Gbps	0.00 Gbps	0.00 Gbps
PlayStation	0.00 Gbps	0.00 Gbps	0.00 Gbps
Nintendo Wii	0.00 Gbps	0.00 Gbps	0.00 Gbps
Naming Services	0.00 Gbps	0.00 Gbps	0.00 Gbps
Terminals	0.08 Gbps	0.01 Gbps	0.00 Gbps
Other Net Admin	0.05 Gbps	0.02 Gbps	0.01 Gbps
Anonimity Networks	0.00 Gbps	0.00 Gbps	0.00 Gbps
Other Well-Known Ports	0.00 Gbps	0.00 Gbps	0.00 Gbps
Windows Live Messenger Video	0.00 Gbps	0.00 Gbps	0.00 Gbps
Web-Based E-Mail	0.00 Gbps	0.00 Gbps	0.00 Gbps
Skype VoIP	0.08 Gbps	0.03 Gbps	0.04 Gbps
Skype File Transfer	0.05 Gbps	0.01 Gbps	0.01 Gbps
RTMP	0.10 Gbps	0.05 Gbps	0.08 Gbps
Other Internet Video	0.02 Gbps	0.01 Gbps	0.01 Gbps
Facebook	0.09 Gbps	0.04 Gbps	0.05 Gbps
MySpace	0.00 Gbps	0.00 Gbps	0.00 Gbps
Twitter	0.00 Gbps	0.00 Gbps	0.00 Gbps
Other Social Sites	0.00 Gbps	0.00 Gbps	0.00 Gbps
MS Exchange Desktop	0.00 Gbps	0.00 Gbps	0.00 Gbps
Facebook IM	0.00 Gbps	0.00 Gbps	0.00 Gbps
OpenVPN	0.02 Gbps	0.00 Gbps	0.00 Gbps
Google Voice	0.00 Gbps	0.00 Gbps	0.00 Gbps
WebEx	0.00 Gbps	0.00 Gbps	0.00 Gbps
FIX	0.00 Gbps	0.00 Gbps	0.00 Gbps
ClickStream-New Page	0.13 Gbps	0.05 Gbps	0.06 Gbps
ClickStream-New Site	0.20 Gbps	0.08 Gbps	0.07 Gbps
Flash YouTube HD	0.00 Gbps	0.00 Gbps	0.00 Gbps
Bittorrent Over IPv6	0.00 Gbps	0.00 Gbps	0.00 Gbps
Flash HD	0.00 Gbps	0.00 Gbps	0.00 Gbps
Viber	0.00 Gbps	0.00 Gbps	0.00 Gbps
Gmail Video	0.00 Gbps	0.00 Gbps	0.00 Gbps
MyPeople	0.00 Gbps	0.00 Gbps	0.00 Gbps
NIX All	1.57 Gbps	0.87 Gbps	1.10 Gbps
NIX P2P	1.21 Gbps	0.66 Gbps	1.00 Gbps
NIX Streaming	1.38 Gbps	0.72 Gbps	0.93 Gbps